P.1877/87

1123

1

1987

informatyka

Prof. Daniel Tabak o Intelu 80386

Sieci Petriego

Lattice/Microsoft C

WYDAWNICTWO NOT SIGMA

Cena 150 zł

informatyka

1 1987

Nr 1

Miesięcznik Rok XXII

Styczeń 1987

Organ Komitetu Informatyki MNSZWiT oraz Komitetu Naukowo-Technicznego NOT ds. Informatyki

KOLEGIUM REDAKCYJNE:

Dr inż. Wacław ISZKOWSKI, mgr Teresa JABŁOŃSKA (sekretarz redakcji), Władysław KLEPACZ (redaktor naczelny), dr inż. Marek MACHURA, Maria PAWLAK (sekretarz redakcji), dr inż. Wiktor RZECZKOWSKI, mgr inż. Jan RYŻKO, mgr Hanna WŁODARSKA, dr inż. Janusz ZALEWSKI (zastępca redaktora naczelnego)

PRZEWODNICZĄCY RADY PROGRAMOWEJ:

Prof. dr hab. Juliusz Lech Kulikowski

Materiałów nie zamówionych redakcja nie zwraca

Redakcja: 00-517 Warszawa, ul. Mickiewicza 18 m. 17, tel. 39-14-34

Zakł. Graf. „Tamka". Zam. 0893-1300/86. Obj. 4,0 ark. druk. Nakład 7700 egz. P-93.

ISSN 0542-9951, INDEKS 36124

Cena egzemplarza 150 zł
Prenumerata roczna 1800 zł

NACZELNA ORGANIZACJA TECHNICZNA

WYDAWNICTWO SIGMA

CZASOPISM I KSIĄŻEK TECHNICZNYCH

00-950 Warszawa
skrytka pocztowa 1004
ul. Biała 4

W NUMERZE:

	Strona
Intel 80386 i nowe mikroprocesory 32-bitowe (1) *Daniel Tabak*	1
Sieci Petriego (1). Definicja i analiza własności sieci *Marek Gondzio*	5
Systemy wielomikroprocesorowe (2) *Wojciech Cellary*	10
Podejścia do tworzenia schematów konceptualnych *Stanisław Wrycza*	13
Język C — implementacja Lattice/Microsoft *Jan Bielecki*	17
Systemy ekspertowe (2) *Jacek Ambroziak*	19
System operacyjny PC-DOS (4) *Roman Grabowicz, Janusz Zalewski*	21
DYDAKTYKA	24
Program nauczania informatyki w małych uczelniach amerykańskich	
Z KRAJU	26
Programista zawodowy i wartość jego pracy Poprawa efektywności testowania wymienności taśm magnetycznych w systemach Odra 1300	
ZE ŚWIATA	28
Wielodostępne systemy operacyjne dla IBM PC/XT (1)	
TERMINOLOGIA	III okł.
Terminologia lokalnych sieci komputerowych (2)	

W NAJBLIŻSZYCH NUMERACH:

- Witold Łukaszewicz o przechowywaniu wiedzy i niewiedzy w bazach danych
- Jacek Owczarczyk, Maciej Stolarski, Ewa Woźniak o współbieżnych systemach przetwarzania obrazów
- Janusz Skolimowski o specjalizowanym systemie mikrokomputerowym SIGMA-3
- Ryszard Tadeusiewicz o nauczaniu informatyki
- Robert Kowalski o programowaniu logicznym i komputerach nowej generacji
- Andrzej Grzywak i Jerzy Kołodziej o sieciach komputerowych opartych na mikrokomputerach Mera 60

W naszej przeszło 20-letniej działalności redakcyjnej rok ubiegły był pod względem wydawniczym wyjątkowo niepomyślny. Z całą wyrazistością odczuliśmy skutki słabości krajowej bazy poligraficznej. Wydawnicze trudności najbardziej odczuwa prasa naukowo-techniczna, głównie ze względu na problem tzw. utrudnionego składu (teksty nasycone wzorami i nietypowymi znakami graficznymi). Ostry deficyt kadr w wydziałach składu, a także mniejsza rentowność druku wskutek ograniczonych nakładów, to dalsze czynniki przyspieszające dążenia drukarni do rozwiązywania umów na druk tej kategorii wydawnictw. Bezpośrednim skutkiem wspomnianych trudności są wielomiesięczne opóźnienia w wydawaniu naszych tytułów.

Aby radykalnie przełamać impas, wywołujący również przykre efekty psychologiczne i to nie tylko w środowisku czytelniczym, Wydawnictwo SIGMA musiało podjąć drastyczną decyzję znacznego ograniczenia objętości INFORMATYKI w 1986 r. Wątpliwym pocieszeniem jest fakt, że analogiczne ograniczenia dotknęły większość czasopism naszego wydawnictwa, w tym również tytuły o dużym znaczeniu gospodarczym i społecznym.

Dla INFORMATYKI oznaczało to wydanie w ubiegłym roku tylko ośmiu zeszytów (zamiast dwunastu) i to w ograniczonej objętości.

Pragniemy gorąco przeprosić Czytelników za zaistniałą sytuację.

Mamy nadzieję, że w bieżącym roku sytuacja ulegnie poprawie, w związku z przeniesieniem INFORMATYKI do nowej własnej drukarni Wydawnictwa SIGMA. Powinno to nie tylko radykalnie poprawić terminowość druku, ale również, dzięki nowoczesnemu wyposażeniu tego zakładu, dokonać odczuwalnego postępu w szacie edytorskiej czasopisma.

Redakcja

DANIEL TABAK
Electrical and Computer Engineering Department
George Mason University
Fairfax, stan Virginia
USA

Intel 80386 i nowe mikroprocesory 32-bitowe (I)

W pierwszej części artykułu przedstawiono rozwój mikroprocesorów rodziny Intel iAPX 86 oraz opisano architekturę mikroprocesora 80386. W drugiej natomiast omówiono cztery nowe mikroprocesory 32-bitowe innych firm (MC 68020, NS 32332, Z80000 i WE 32100) i porównano je z mikroprocesorem firmy Intel.

ROZWÓJ RODZINY iAPX 86

Pierwszym mikroprocesorem 16-bitowym firmy Intel, wprowadzonym na rynek w 1978 roku, był mikroprocesor 8086 [2, 5]. Był wytwarzany w technologii HMOS i zawierał ok. 29000 tranzystorów. Rozszerzono w nim pamięć bezpośrednio adresowalną od 64 KB (jak w mikroprocesorach 8080 i 8085) do 1 MB, lecz utrzymano zgodność programową z poprzednikami 8-bitowymi. Obecnie działa on przy częstotliwości zegarowej do 10 MHz (8086-1). Stosując multipleksowane linie danych i linie adresowe (AD0-AD15), utrzymano 40-końcówkową obudowę mikroukładu. Pamięć tego mikroprocesora jest segmentowana, podzieloną na stałe 64-kilobajtowe segmenty (maksymalną pojemność pamięci mikroprocesorów 8-bitowych). Segment może rozpoczynać się od dowolnego adresu pamięci, podzielnego przez 16 (tzn. cztery najmniej znaczące bity adresu bazowego muszą równać się zeru), a sąsiednie segmenty mogą zachodzić na siebie.

Architektura mikroprocesora 8086 składa się z ośmiu 16-bitowych rejestrów roboczych. Nie wszystkich z nich można używać w dowolny sposób, niektóre mają ściśle określone funkcje. Cztery rejestry danych: **AX, BX, CX i DX** są dość uniwersalne, za wyjątkiem użycia w niektórych operacjach arytmetycznych i wejścia-wyjścia. Cztery rejestry wskaźnikowe: **SP** (ang. stack pointer, wskaźnik stosu), **BP, SI,** i **DI** służą jako wskaźniki oraz rejestry bazowe i indeksowe w różnych trybach adresowania. Rejestry **AX-DX** są podzielone wewnętrznie na bajty (**AX=AH+AL** itd.), aby utrzymać zgodność z architekturą mikroprocesora 8080, tzn. **AL=A, BH=H, BL=L, CH=B, CL=C, DH=D** i **DL=E.** Licznik rozkazów nazywa się w mikroprocesorze 8086 wskaźnikiem rozkazów (ang. instruction pointer, IP) i jest oddzielnym rejestrem 16-bitowym. Ponadto istnieją cztery 16-bitowe rejestry segmentowe:

CS — segmentu kodu
SS — segmentu stosu
DS — segmentu danych
ES — segmentu dodatkowego

przechowujące górne 16 bitów adresu bazowego odpowiedniego segmentu. Adres w mikroprocesorze 8086 tworzy się przesuwając zawartość rejestru segmentu o 4 bity w lewo i dodając 16-bitowe wyrównanie (ang. offset), wskutek czego otrzymuje się 20-bitowy adres fizyczny. Wyrównanie uzyskuje się zależnie od użytego trybu adresowania.

Mikroprocesor 8086 ma dziesięć podstawowych typów adresowania:

- rejestrowy
- prosty (ang. immediate)
- bezpośredni
- pośredni rejestrowy
- bazowy
- indeksowy
- indeksowy bazowy
- bazowy z przemieszczeniem (ang. with displacement)
- indeksowy z przemieszczeniem
- indeksowy bazowy z przemieszczeniem.

W celu zapewnienia pełnej zgodności programowej mikroprocesora 16-bitowego z 8-bitowym o tej samej archi-

c2.14.007

tekturze, a także aby umożliwić używanie tego mikroprocesora w konstrukcjach opartych na mikroprocesorach 8080 i 8085, w firmie Intel opracowano mikroprocesor 8088. Ma on budowę wewnętrzną prawie identyczną jak 8086, a główna różnica polega na utrzymaniu jedynie ośmiu linii danych (tzn. końcówki A8-A15 służą tylko do przesyłania adresów). Ponadto mikroprocesor 8088 ma tylko 4-bajtową kolejkę rozkazową, w przeciwieństwie do 6-bajtowej kolejki mikroprocesora 8086. Oczywiście, ponieważ procesor 8088 pobiera na raz tylko jeden bajt, jest znacznie wolniejszy od 8086, który pobiera dwa bajty jednocześnie. Jednakże zastosowano go w komputerach osobistych IBM PC i wielu innych. Mikroprocesory 8086 i 8088 opatrzono później nazwami iAPX 86 i iAPX 88.

Do współpracy z mikroprocesorami 8086 i 8088 są przeznaczone, działające z nimi równolegle, tzw. koprocesory:

- koprocesor arytmetyczny (zmiennoprzecinkowy) 8087
- koprocesor wejścia-wyjścia 8089.

Dalszym rozszerzeniem tej rodziny mikroprocesorów było opracowanie mikromodułu 80186 (iAPX 186). Jest to układ VLSI wykonany w technologii HMOS II, mający obudowę 68-końcówkową. Jego architektura jest prawie identyczna jak mikroprocesora 8086, choć dodano mu 10 nowych rozkazów. Istnieje pełna wstępująca zgodność kodu wynikowego między mikroprocesorami 8086 i 80186. Podstawowa różnica w organizacji obu mikroprocesorów polega na tym, że w mikroukładzie 80186 umieszczono dodatkowe podsystemy, tj.:

— generator zegarowy o częstotliwości 8 MHz
— dwa niezależne kanały DMA o dużej szybkości
— programowany sterownik przerwań
— trzy programowane, 16-bitowe czasomierze
— programowaną logikę do wyboru pamięci i mikroukładów
— programowany generator stanu WAIT
— sterownik magistrali lokalnej.

Umieszczenie wymienionych podsystemów w pojedynczym mikroukładzie eliminuje potrzebę użycia dużej liczby układów zewnętrznych, w systemach opartych na mikroprocesorze 80186.

Podobnie jak dla mikroprocesora 8086, wprowadzono także mikroprocesor z 8-bitową zewnętrzną szyną danych, 80188, pod innymi względami identyczny z procesorem 80186.

Znaczne rozszerzenia architektury mikroprocesora 8086 zrealizowano w następnym mikroukładzie tej rodziny — procesorze 80286 (iAPX 286). Zrealizowano go jako układ o częstotliwości zegarowej 10 MHz, zawierający 130 000 tranzystorów, umieszczony w obudowie 68-końcówkowej. Zasadnicze rozszerzenia wprowadzone w tym mikroprocesorze są związane z wieloprogramowością i wielozadaniowością oraz z mechanizmami ochrony. Utrzymano przy tym zgodność programową (wstępującą, ang. upward) z poprzednimi mikroprocesorami tej rodziny (8086, 8088, 80186 i 80188).

Mikroprocesor 80286 ma dwa podstawowe tryby pracy:
- tryb rzeczywisty z procesorem iAPX 86 (ang. iAPX 86 Real Address Mode), umożliwiający wykonywanie rozszerzonej listy rozkazów mikroprocesora 8086 i dostęp do pamięci operacyjnej o pojemności 1 MB;
- tryb ochrony (ang. protected virtual address mode, PVAM), umożliwiający pracę wielozadaniową w środowisku chronionym i dostęp do pamięci fizycznej o pojemności 16 MB i pamięci wirtualnej 1 GB dla każdego zadania.

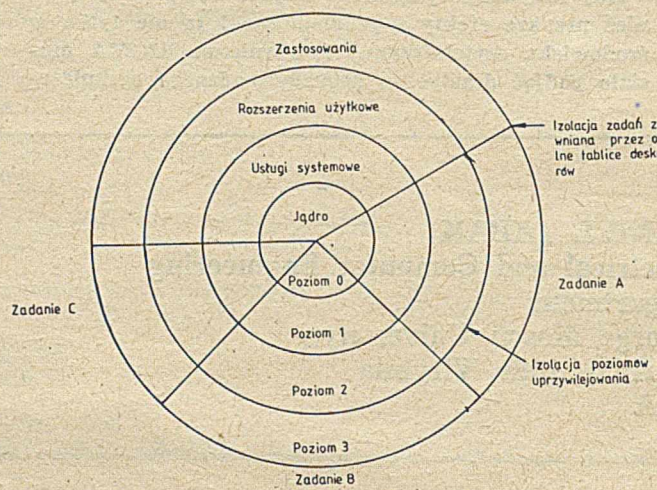
W rezultacie mikroprocesor 80286 stanowi pomost między pierwotnym rozwiązaniem 8086 a najnowszym — 80386, opisanym w następnych punktach. W mikroprocesorze 80286 zastosowano także pamięć segmentową (tj. zarządzanie pamięcią w mikroukładzie), jednak w przeciwieństwie do podziału na segmenty o stałej długości 64 KB, jak w mikroprocesorze 8086, długość segmentu może się zmieniać od 1 B do 64 KB. Oprócz rejestrów mikroprocesora 8086, w procesorze 80286 umieszczono rejestry dodatkowe:

- cztery 48-bitowe rejestry podręczne segmentu (ang. segment cache registers) zawierające adres bazowy, ograniczenie rozmiaru i prawa dostępu dla czterech rejestrów segmentowych (CS, SS, DS i ES),
- cztery 48-bitowe wskaźniki segmentu tablicy i zadania (ang. table and task segment pointers), zawierające ten sam rodzaj informacji co rejestry podręczne segmentu,
- 16-bitowy rejestr tablicy deskryptorów lokalnych (ang. local descriptor table register, LDTR),
- 16-bitowy rejestr zadań (ang. task register, TR),
- 16-bitowy rejestr słowa stanu maszyny (ang. machine status word, MSW).

W porównaniu z mikroprocesorem 8086, znacznie rozszerzono listę rozkazów, dodając m.in. 15 nowych rozkazów do sterowania ochroną kodu, stosowanych w trybie PVAM. Wprowadzono także cztery poziomy ochrony (rys. 1):

0 — jądro systemu operacyjnego (poziom najbardziej uprzywilejowany),
1 — usługi systemowe,
2 — użytkowe rozszerzenia systemu operacyjnego,
3 — zastosowania (poziom najmniej uprzywilejowany).

Im mniejsza jest wartość liczbowa poziomu, tym większe jest jego uprzywilejowanie. Tę samą strukturę poziomów uprzywilejowania utrzymano w mikroprocesorze 80386. W rzeczywistości, wiele cech mikroprocesora 80286 zachowano w rozwiązaniu mikroukładu 80386, niekiedy je rozszerzając.



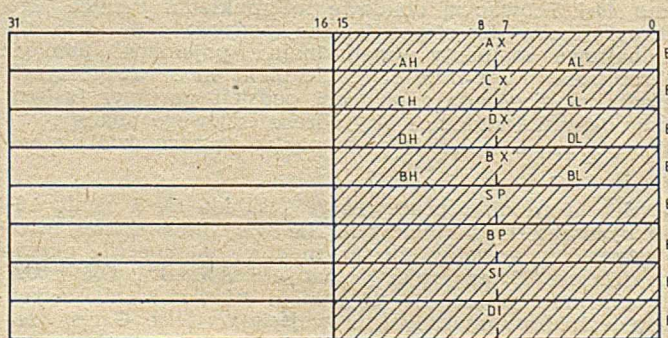
Rys. 1. Poziomy ochrony kodu w mikroprocesorach Intel 80286 i 80386 (wszystkie rysunki reprodukowano za zgodą Intel Corp.)

W mikroprocesorze 80286, 16 końcówek obudowy odpowiadających liniom D0-D15 całkowicie oddzielono od 24 końcówek odpowiadających liniom adresowym A0-A23. Ponadto, oprócz 6-bajtowej kolejki wstępnie pobranych rozkazów, istniejącej także w mikroprocesorze 8086, wprowadzono dodatkowo 3-elementową kolejkę rozkazów zdekodowanych. Mikroprocesor 80286 ma trzy jednostki arytmetyczno-logiczne:
- normalną jednostkę do przetwarzania danych
- sumator wyrównania (ang. offset adder)
- sumator adresu fizycznego (ang. physical address adder).

Wymienione rozszerzenia zwiększają przepustowość mikroprocesora 80286, czyniąc go ok. 6 razy szybszym od procesora 8086.

ARCHITEKTURA MIKROPROCESORA 80386

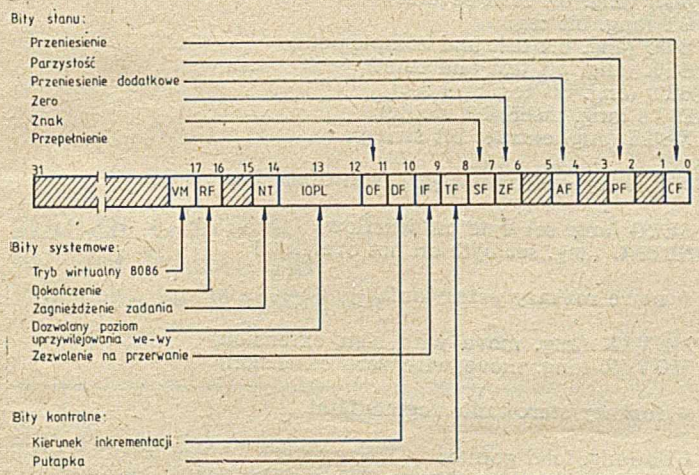
Mikroprocesor 80386 ma osiem 32-bitowych rejestrów roboczych (rys. 2). Zakreskowane, 16-bitowe części tych rejestrów odpowiadają dokładnie rejestrom roboczym starszych mikroprocesorów tej rodziny (8086, 8088, 80186, 80188, 80286). W mikroprocesorze 80386 wszystkie te rejestry rozszerzono do 32 bitów i zmieniono ich nazwy, dodając przed-



Rys. 2. Rejestry robocze mikroprocesora Intel 80386

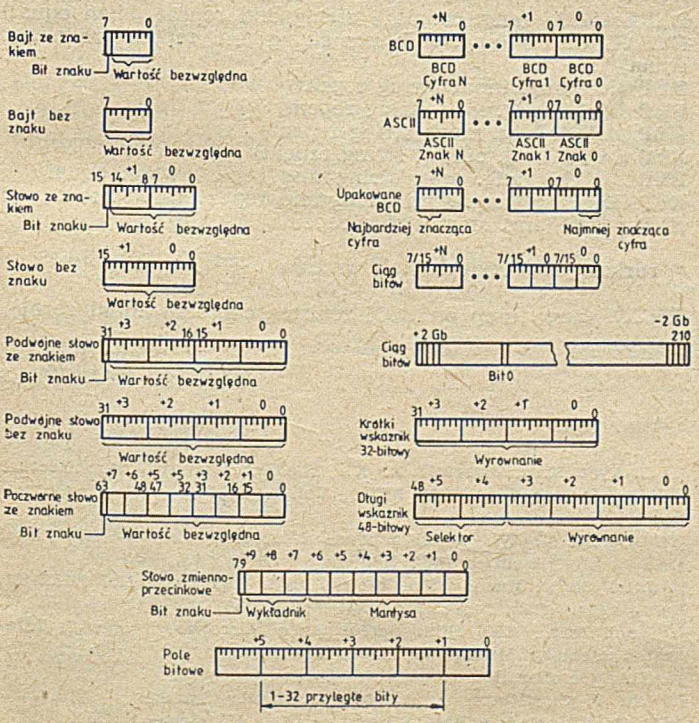
rostek E (ang. extended). Części 8- i 16-bitowe tych rejestrów zachowały swoje pierwotne nazwy. W mikroprocesorze 80386 utrzymano również ogólny podział funkcji między te rejestry [3].

Mikroprocesor 80386 ma sześć 16-bitowych rejestrów wyboru segmentu (ang. segment selector registers): CS, SS, DS, ES, FS i GS. Starsze mikroprocesory tej rodziny zawierały tylko cztery z nich, a rejestry FS i GS wprowadzono dopiero w procesorze 80386. Tak więc, program wykonywany na tym mikroprocesorze może odwoływać się do sześciu segmentów. Z każdym rejestrem wyboru segmentu jest skojarzony 64-bitowy rejest deskryptorowy (w mikroprocesorze 80286 są to rejestry 48-bitowe), zawierający fizyczny adres bazowy, ograniczenie (rozmiar) segmentu oraz inne atrybuty tego segmentu. Podobnie jak w mikroprocesorach 8086 i 80286 rejestr CS wskazuje aktualnie używany segment kodu, rejestr SS — segment stosu, natomiast rejestry DS, ES, FS i GS reprezentują segmenty danych.



Rys. 3. Rejestr EFLAGS mikroprocesora 80386

Ponadto istnieje 32-bitowy wskaźnik rozkazów (EIP) oraz 32-bitowy rejestr stanu (EFLAGS), zawierające odpowiednio 16-bitowe rejestry IP i FLAGS mikroprocesora 80286. Dokładne rozmieszczenie bitów w rejestrze EFLAGS przedstawiono na rys. 3. Bity 0—15 mają znaczenie identyczne jak w rejestrze FLAGS mikroprocesora 80286. Spośród bitów części bardziej znaczącej używane są jedynie bity 16 i 17. Bit 16, RF (ang. resume flag, znacznik dokończenia), ma znaczenie dla punktów wstrzymania (ang. breakpoints) przy uruchamianiu programów lub w pracy krokowej. Jego ustawienie powoduje zignorowanie defektu w następnym rozkazie uruchamianego kodu. Bit 17, VM (ang. virtual 8086 mode) służy do przełączania mikroprocesora 80386 z trybu ochrony (ang. protected mode) na pracę w trybie wirtualnym mikroprocesora 8086 [4].

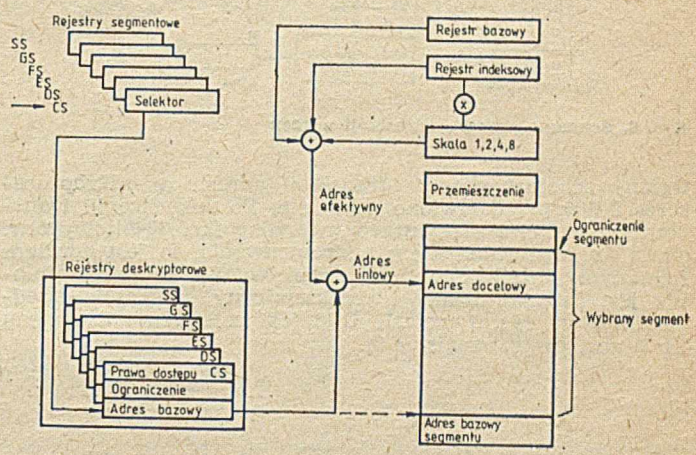


Rys. 4. Typy danych mikroprocesora 80386

Na rys. 4 przedstawiono typy danych [1] występujące w mikroprocesorze 80386, stosowane powszechnie w językach wysokiego poziomu. Trzy formaty liczb zmiennoprzecinkowych (o długości 32, 64 i 80 bitów) są zapewnione przez koprocesory arytmetyczne 80287 lub 80387.

Podstawowe tryby adresowania w mikroprocesorach tej rodziny rozszerzono do 11 trybów mikroprocesora 80386:

- rejestrowy
- prosty
- bezpośredni
- pośredni rejestrowy
- bazowy
- indeksowy
- indeksowy skalowany
- indeksowy bazowy
- bazowy indeksowy skalowany
- indeksowy bazowy z przemieszczeniem
- bazowy indeksowy skalowany z przemieszczeniem.



Rys. 5. Obliczanie adresów przy różnych trybach adresowania

Zasady obliczania adresów przy różnych trybach adresowania przedstawiono na rys. 5. Główna zmiana w porównaniu z adresowaniem w mikroprocesorach 8086 i 80286 polega na wprowadzeniu skalowania (o skali równej 1, 2, 4 lub 8). Ponadto istnieje większa dowolność wyboru rejestrów bazowych i indeksowych, którymi mogą być dowolne 32-bitowe rejestry robocze (tab.).

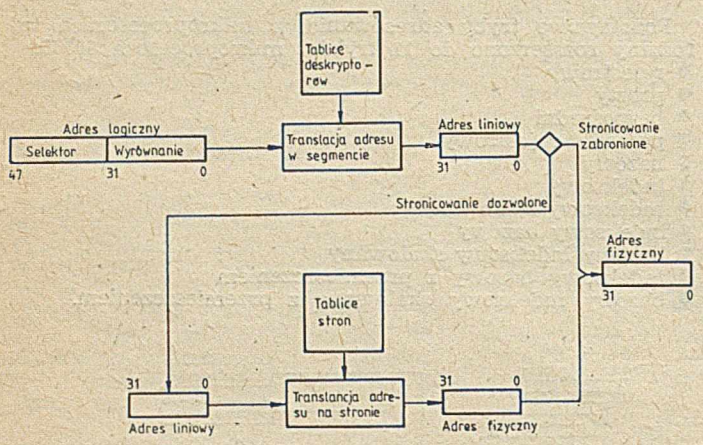
Porównanie możliwości adresowania 16-bitowego i 32-bitowego

	Adresowanie 16-bitowe	Adresowanie 32-bitowe
Rejestry bazowe	BX, BP	Dowolny 32-bitowy rejestr roboczy
Rejestry indeksowe	SI, DI	Dowolny 32-bitowy rejestr roboczy (oprócz ESP)
Skala	—	1, 2, 4, 8
Przemieszczenie	0, 8, 16 bitów	0, 8, 32 bity

W mikroprocesorze 80386 rozszerzono znacznie przestrzeń adresową pamięci. Fizyczna przestrzeń adresowa tego mikroprocesora ma pojemność 4 GB (2^{32} B), a więc znacznie więcej od 16-megabajtowej i 1-megabajtowej pamięci mikroprocesorów 80286 i 8086. Wirtualna (logiczna) przestrzeń adresowa dla zadania wynosi 64 terabajty (2^{46} B). Dopuszczalna jest także segmentacja pamięci, lecz górne ograniczenie ma rozmiar segmentu rozszerzonego do 4 GB, tj. do maksymalnej pojemności fizycznej pamięci. Tak więc, długość segmentu w mikroprocesorze 80386 może wynosić od 1 B do 4 GB. Wybór długości segmentu równej 4 GB jest równoważny pominięciu segmentacji. W ten sposób projektanci mikroprocesora 80386 uniknęli jednej z głównych wad starszych mikroprocesorów tej rodziny, tj. ograniczenia roz-

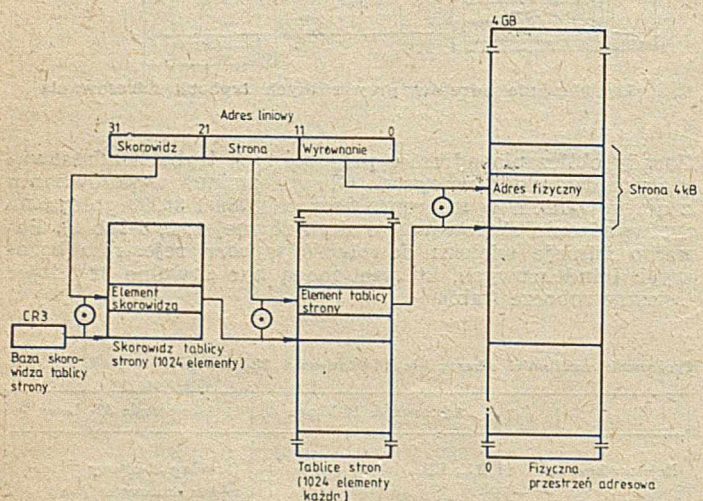
miaru segmentu do 64 KB. Procesor 80386 ma również opcjonalny mechanizm stronicowania pamięci o ustalonej długości 4 KB. Z punktu widzenia organizacji pamięci mikroprocesor 80386 może być używany na cztery sposoby:

- bez segmentacji i stronicowania (tzw. pamięć liniowa)
- z segmentacją pamięci (zmienne rozmiary segmentów)
- ze stronicowaniem pamięci (ustalone rozmiary stron — 4 KB)
- z segmentacją i stronicowaniem pamięci jednocześnie.



Rys. 6. Schemat procesu translacji adresu

Ogólny schemat procesu translacji adresu w mikroprocesorze 80386 przedstawiono na rys. 6. Po zakończeniu translacji związanej z segmentacją tworzony jest 32-bitowy tzw. adres liniowy. Jeżeli nie ma stronicowania, to jest adresem fizycznym. Jeżeli występuje stronicowanie, to adres liniowy jest interpretowany i przekształcany w sposób przedstawiony na rys. 7 (rejestr CR3 jest rejestrem sterującym nr 3, ang. control register 3).



Rys. 7. Przekształcanie adresu liniowego na fizyczny przy występowaniu stronicowania

Pamięć w mikroprocesorze 80386 jest adresowana bajtowo. Słowa 16-bitowe są dosuwane do granic o adresach parzystych, a słowa podwójne (32-bitowe) — do granic o adresach podzielnych przez 4. W słowie pojedynczym lub podwójnym najmniej znaczący bajt ma najniższy adres, identyczny z adresem całego słowa.

W mikroprocesorze 80386 przestrzeń wejścia-wyjścia jest oddzielona od przestrzeni pamięci, choć dopuszczalne jest także wejście-wyjście odwzorowane w pamięci. Pełna przestrzeń we-wy ma 64 KB pojemności i może być podzielona w następujący sposób: 64 K portów 8-bajtowych, 32 K portów 16-bajtowych i 16 K portów 32-bajtowych. Dozwolona jest także dowolna kombinacja wymienionych podziałów, do łącznej pojemności 64 KB. Dostęp do portów we-wy jest możliwy za pomocą instrukcji IN i OUT. Adresy portów są dostarczane w rejestrach DL, DX i EDX. Wszystkie adresy portów 8- i 16-bitowych są uzupełniane zerami na bardziej znaczących liniach szyny adresowej.

Listę rozkazów mikroprocesora 80386 można podzielić na 9 kategorii:

— przesyłanie danych (32 rozkazy)
— rozkazy arytmetyczne (17)
— operowanie napisami (12)
— rozkazy logiczne, przesunięcia i obroty (15)
— operowanie bitami (8)
— przekazywanie sterowania (45)
— wspomaganie języków wysokiego poziomu (3)
— rozkazy pracy w trybie ochrony (15)
— sterowanie procesorem (4).

Łącznie mikroprocesor ma 151 rozkazów. Poszczególne rozkazy mogą mieć 0—3 argumenty o długości 8, 16 lub 32 bity. W mikroprocesorze 80386 zachowano podstawową listę rozkazów procesora 80286, dodając jednak wiele nowych:

- **rozkazy operowania bitami**

BSF (ang. bit scan forward)
BSR (ang. bit scan reverse)
BT (ang. bit test)
BTC (ang. bit test and complement)
BTR (ang. bit test and reset)
BTS (ang. bit test and set)
IBTS (ang. insert bit string)
XBTS (ang. extract bit string).

- **rozkazy warunkowego ustawiania bajtów**

SETO (ang. set byte on overflow)
SETNO (ang. set byte on not overflow)

- **nowe rozkazy przesyłania**

MOVSX (ang. move with sign extension)
MOVZX (ang. move with zero extension)

- **rozkazy sterowania segmentami**

LFS (ang. load pointer to FS register)
LGS (ang. load pointer to GS register)
LSS (ang. load pointer to SS register)

- **rozkaz sterowania znacznikami**

CLTS (ang. clear task switched flag)

- **rozkazy operacji logicznych**

SHLD (ang. shift left double)
SHRD (ang. shift right double)

- **rozkazy powtarzalnego operowania napisami z odliczaniem w rejestrze CX**

REPE CMPS (ang. compare string and repeat when equal)
REPNE CMPS (ang. compare string and repeat when not equal)
REP INS (ang. input string repeat)
REP LODS (ang. load string repeat)
REP MOVS (ang. move string repeat)
REP OUTS (ang. output string repeat)
REPE SCAS (ang. scan string and repeat when equal)
REPNE SCAS (ang. scan string and repeat when not equal)
REP STOS (ang. store string repeat)

- **rozkaz przekazywania sterowania**

JECXZ (ang. jump on ECX register contents equal zero)

Tłum. i oprac.
JANUSZ ZALEWSKI

LITERATURA

[1] El-Ayat K.A., Agarwal R. K.: The Intel 80386 — Architecture and Implementation. IEEE Micro, Vol. 5, No. 6, pp. 4—22, December 1985
[2] Gorsline G. W.: 16-Bit Modern Microcomputers — The Intel 18086 Family. Prentice Hall, Englewood Cliffs (NJ), 1985
[3] Intel Corp.: Introduction to the 80386. Document No. 231252-001, Santa Clara (CA), September 1985
[4] Intel Corp.: 80386 High Performance Microprocessor with Integrated Memory Management. Document No. 231630-001, Santa Clara (CA), October 1985
[5] Liu Y. C., Gibson G. A.: Microcomputer Systems — The 8086/8088 Family (2nd edition). Prentice Hall, Englewood Cliffs (NJ), 1986.

Sieci Petriego — studenci nazywają je grą w chińczyka. Formalnie sieć Petriego jest specyficznym grafem służącym do opisu asynchronicznie funkcjonujących zjawisk, a w zakresie naszych zainteresowań — głównie w oprogramowaniu. Sam autor sieci — C. A. Petri, podając ich teorię i zastosowanie do badania funkcjonowania centrali telekomunikacyjnej, nie przewidział, że jego pomysł zapoczątkuje ogromną lawinę badań, opracowań naukowych i zastosowań. Niewątpliwie sieci Petriego pozwoliły na uporządkowanie wielu pojęć z zakresu badania zależności pomiędzy zdarzeniami zachodzącymi równocześnie w wielu obiektach. Z drugiej strony, w wielu gabinetach uczonych, zajmujących się teorią sieci, wiszą ogromne grafy zbudowane z kółek, kresek i krawędzi o takim stopniu złożoności, że ich analiza w sensownym czasie — nawet z udziałem komputera — jest niemożliwa. Stąd pojawiło się pewne rozczarowanie tym pomysłem. Sieci Petriego są jednak tak znaczącym elementem w teorii informatyki, że warto przypomnieć najważniejsze zasady ich funkcjonowania oraz różne kierunki ich modyfikacji.

Celem artykułu jest wyjaśnienie podstawowych pojęć związanych z sieciami Petriego, co powinno pozwolić na samodzielne studiowanie bogatej literatury na ten temat. W pierwszej części artykułu zdefiniowano terminy spotykane w opisie i analizie sieci oraz podano interpretację wprowadzonych pojęć, ilustrując je przykładami modeli sieciowych. Przedstawiono również metody badania typowych własności sieci.

W drugiej części artykułu zostaną przedstawione podstawowe przekształcenia oraz główne rodzaje ograniczonych i rozszerzonych sieci Petriego. (Red.)

MAREK GONDZIO
Instytut Informatyki
Politechnika Warszawska

Sieci Petriego (1)

Definicja i analiza własności sieci

Mija już prawie ćwierć wieku od ogłoszenia pierwszych prac C. A. Petriego [15] zawierających definicję sieci nazwanych później jego imieniem. Znaczący rozwój teorii tych sieci nastąpił w latach siedemdziesiątych, równolegle z szybkim rozwojem programowania współbieżnego, gdyż właśnie w zakresie zjawisk współbieżnych znalazły one najwięcej zastosowań. Najważniejsze osiągnięcia w pracach teoretycznych i badawczych są udziałem naukowców skupionych w takich ośrodkach, jak MIT (Cambridge, Massachusetts, USA), LAAS (Tuluza), GMD (Bonn), a także uniwersytetów w Aarhus (Dania), Grenoble, Paryżu i Mediolanie. Szczególnie dużą popularność zyskały sieci Petriego w Europie. Liczne prace na ten temat prowadzono również w NRD, ZSRR [21] i w Polsce.

Teoria sieci Petriego stanowi dziedzinę otwartą, rozwijaną w różnych kierunkach, czego potwierdzenie można znaleźć w materiałach wielu konferencji naukowych. Prezentacji i konfrontacji osiągnięć służy m.in. odbywająca się od 1980 roku coroczna konferencja pod nazwą: European Workshop on Applications and Theory of Petri Nets. Również lista praktycznych zastosowań sieci Petriego jest ciągle wzbogacana o nowe pozycje.

Mgr inż. MAREK GONDZIO ukończył w 1979 r. Wydział Elektroniki Politechniki Warszawskiej. Obecnie jest starszym asystentem w Instytucie Informatyki PW. Zajmuje się oprogramowaniem mikrokomputerów oraz metodami opisu i weryfikacji asynchronicznej współpracy zespołów cyfrowych.

Siecią Petriego (ang. Petri Net, ozn. PN) nazywa się zwykle [6] trójkę $PN = \langle P, T, S \rangle$, w której:

P — jest niepustym, skończonym zbiorem miejsc (ang. places) sieci,

T — jest niepustym, skończonym zbiorem przejść (ang. transitions) sieci,

A — jest niepustym zbiorem skierowanych krawędzi (łuków) łączących miejsca z przejściami oraz przejścia z miejscami:

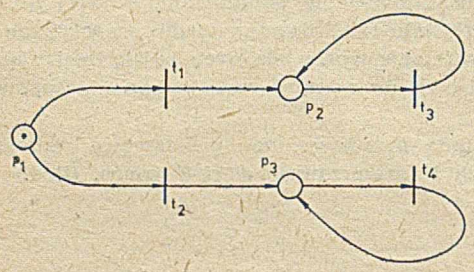
$$A \subseteq P \times T \cup T \times P$$

$\text{dom}(A) \cup \text{cod}(A) = P \cup T$, gdzie dom(A) i cod(A) oznaczają odpowiednio dziedzinę i przeciwdziedzinę relacji A. Ostatni warunek wyraża fakt, że każdy z elementów zbiorów P i T musi wystąpić w relacji A, czyli, że z każdym miejscem i z każdym przejściem musi być związana jakaś krawędź. Niektóre definicje sieci Petriego inaczej formułują ten warunek, np. w [17, 19] przyjmuje się dodatkowo, że dla każdego przejścia musi istnieć co najmniej jedna krawędź wejściowa (tzn. skierowana „do przejścia") i co najmniej jedna krawędź wyjściowa. Wszystkie przykłady sieci Petriego zamieszczone w artykule spełniają ten rozszerzony warunek. Przy graficznym przedstawieniu sieci Petriego miejsca są oznaczone małymi okręgami, a przejścia prostokątami lub pogrubionymi kreskami.

Znakowaniem (ang. marking) M sieci PN jest funkcja $M : P \to N$, $N = \{0, 1, 2, ...\}$ przyporządkowująca każdemu miejscu nieujemną liczbę całkowitą, określającą liczbę znaczników (ang. tokens) znajdujących się w danym miejscu. Graficznie znaczniki są reprezentowane zazwyczaj kropkami.

Siecią znakowaną (ang. Marked Petri Net, MPN) nazywa się parę $MPN = \langle PN, M_0 \rangle$, której PN to sieć Petriego, a M_0 jest znakowaniem początkowym (ang. initial marking). Zbiór P jest z definicji skończony, a więc każde znakowanie może być przedstawione za pomocą wektora. $M = [m_1, m_2, \ldots m_n]$, gdzie $n = \text{card}(P)$ a $m_1, m_2, \ldots m_n$ reprezentują liczbę znaczników odpowiednio w miejscach $p_1, p_2, \ldots p_n$. Często dla oznaczenia liczby znaczników w miejscu p używa się także zapisu M(p).

Przykład prostej sieci znakowanej jest przedstawiony graficznie na rys. 1. Na sieć tę składają się 3 miejsca (p_1, p_2 i p_3), cztery przejścia (t_1, t_2, t_3 i t_4) oraz 8 krawędzi. W znakowaniu początkowym tej sieci występuje tylko jeden znacznik (w miejscu p_1), tzn. $M_0(p_1) = 1$, $M_0(p_2) = 1$, $M_0(p_3) = 0$, czyli $M_0 = [1, 0, 0]$.



Rys. 1. Sieć znakowana MPN_1

W omawianiu sieci Petriego przydatne są pewne terminy pomocnicze. Często zamiast pełnej nazwy **„znakowana sieć Petriego"** mówi się krócej **„sieć znakowana"**, **„sieć ze znakowaniem"** lub po prostu **„sieć"**, gdy z kontekstu wynika już, że chodzi o MPN. Przyjmuje się także następujące oznaczenia pomocnicze:

$^{\cdot}t = \{p \in P: \langle p, t \rangle \in A$ — zbiór miejsc wejściowych przejścia t,

$t^{\cdot} = \{p \in P: \langle t, p \rangle \in A$ — zbiór miejsc wyjściowych przejścia t,

$p^{\cdot} = \{t \in T: \langle p, t \rangle \in A$ — zbiór przejść wyjściowych miejsca p,

$^{\cdot}p = \{t \in T: \langle t, p \rangle \in A$ — zbiór przejść wejściowych miejsca p,

Na przykład — w sieci MPN_1 (rys. 1) $p_1 = \Phi$, $p_1^{\cdot} = \{t_1, t_2\}$, $^{\cdot}t_3 = t_3^{\cdot} = \{p_3\}$.

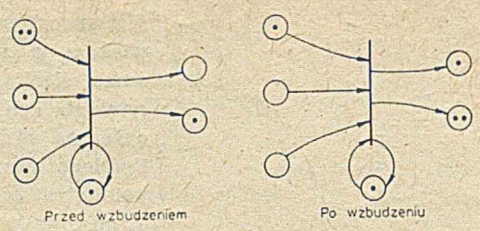
Używając tej notacji można już łatwo i precyzyjnie zdefiniować podstawowe, elemntarne zdarzenie, jakie może zachodzić w sieci znakowanej, czyli **wzbudzenie** (ang. firing) przejścia. Wzbudzone może być tylko przejście przygotowane. **Przejście** t sieci MPN jest **przygotowane** (ang. fireable) przez znakowanie M (lub w znakowaniu M) wtedy i tylko wtedy, gdy każde miejsce wejściowe przejścia t ma co najmniej 1 znacznik: $\forall p \in {}^{\cdot}t: M(p) > 0$.

W sieci z rys. 1, w znakowaniu M_0 przygotowane są przejścia t_1 i t_2.

Każde przejście przygotowane przez znakowanie M może zostać wzbudzone, wyznaczając nowe znakowanie sieci. **Wzbudzenie przejścia** t, przygotowanego przez znakowanie M, polega na jednoczesnym usunięciu po jednym znaczniku z każdego miejsca wejściowego tego przejścia oraz na jednoczesnym wprowadzeniu po jednym znaczniku do każdego miejsca wyjściowego tego przejścia. Wzbudzenie przejścia wyznacza nowe znakowanie M' zdefiniowane następująco:

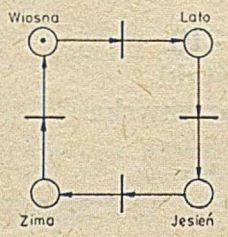
$$\forall p \in P: M'(p) = \begin{cases} M(p) - 1, & \text{jeżeli } p \in {}^{\cdot}t - t^{\cdot}, \\ M(p) + 1, & \text{jeżeli } p \in t^{\cdot} - {}^{\cdot}t, \\ M(p), & \text{jeżeli inaczej.} \end{cases}$$

Wzbudzenie przejścia może powodować nie tylko zmianę położenia znaczników, ale także zmianę ich liczby. Gdy liczba miejsc wejściowych jest mniejsza od liczby miejsc wyjściowych, to występuje **generacja znaczników.** Natomiast, gdy liczba miejsc wejściowych przewyższa liczbę miejsc wyjściowych zachodzi **pochłanianie znaczników.** Ten drugi przypadek ilustruje rys. 2. Dla precyzyjnego zdefiniowania reguł wyznaczających ruch znaczników w sieci — konieczne jest określenie, co zachodzi w sytuacji, gdy w danym znakowaniu liczba przejść przygotowanych jest większa od 1. Taki przypadek występuje w znakowaniu początkowym sieci MPN_1, w którym przygotowane są przejścia t_1 i t_2. Przejścia takie nazywa się **konfliktowymi** —



Rys. 2. Wzbudzenie przejścia

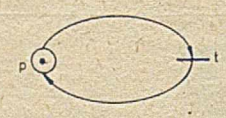
ponieważ wzbudzenie jednego z nich wycofuje przygotowanie drugiego. W takiej sytuacji wybór przejścia do wzbudzenia jest dowolny (losowy). Natomiast innego rozstrzygnięcia wymaga problem, gdy przejścia przygotowane przez dane znakowanie nie są konfliktowe. W dalszej części artykułu przyjmuje się, że wzbudzone zostanie tylko jedno dowolnie wybrane przejście, chociaż spotyka się także i inne podejście [9], dopuszczające — z pewnymi ograniczeniami — jednoczesne wzbudzenie kilku przejść, co jest traktowane jako pojedynczy krok w ruchu znaczników w sieci.



Rys. 3. Sieciowy model zmian pór roku

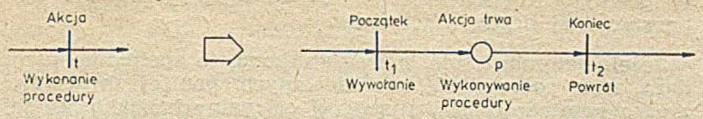
Wzbudzenie przejścia, zmieniając znakowania zmienia także bieżący stan sieci. Oznacza to, że sieci znakowane mogą służyć do prezentacji i modelowania pewnych zjawisk zachodzących w czasie. Jest to podstawowe przeznaczenie sieci Petriego. Rodzaje zjawisk, jakie mogą być modelowane za pomocą sieci Petriego są różnorodne i zależą jedynie od interpretacji przypisywanej miejscom, przejściom, znacznikom czy wzbudzeniom przejść. Jeżeli przyjąć, że miejsca reprezentują pewne warunki, przejścia — zdarzenia, obecność znacznika w danym miejscu — spełnienie odpowiedniego warunku, a wzbudzenie przejścia wyraża zajście pewnego zdarzenia, to sieci Petriego można stosować do modelowania **systemów zdarzeń warunkowych.** Przejścia z reguły reprezentują pewne akcje, a miejsca najczęściej oznaczają pewne stany. Przy różnej interpretacji nawet takie same sieci mogą obrazować różne zjawiska. Klasycznym już tego przykładem jest sieć z rys. 3, przedstawiająca model zmian pór roku (zmianę pory roku reprezentuje tu wzbudzenie przejścia). Jednocześnie tę samą sieć można uznać za model 4-fazowego zegara. Nawet najprostszej sieci (rys. 4) można praktycznie nadać różne znaczenia. Oto niektóre z nich podane przez C. A. Petriego w [16]:

w chemii: p jest katalizatorem reakcji t,
w logice: t jest tautologią „$p \Rightarrow p$",
w informatyce: t oznacza kopiowanie danej p (np. odświeżanie pamięci poprzez odczyt).



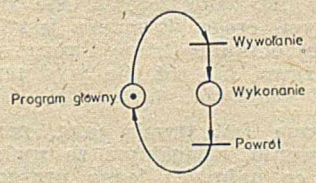
Rys. 4. Najprostsza sieć Petriego

Zgodnie z definicją — wzbudzenie przejścia jest operacją natychmiastową (inaczej mówiąc, czas trwania wzbudzenia jest równy 0). Gdy więc chcemy modelować za pomocą sieci Petriego trwanie pewnych akcji, stosujemy sposób pokazany na rys. 5. Może to być np. model wykonania pewnej procedury, w którym przejścia t_1 i t_2 reprezentują operacje wywołania i powrotu z procedury, a sama czynność wykonania procedury jest reprezentowana przez obecność znacznika w miejscu p.

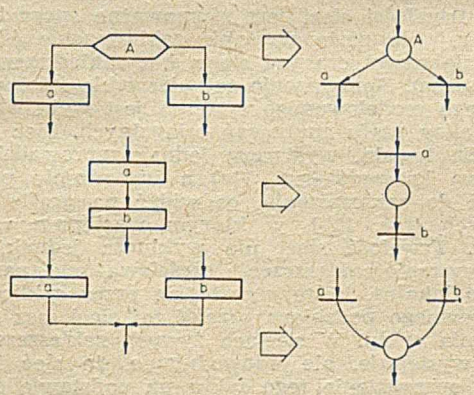


Rys. 5. Modelowanie trwania akcji

Nie ma tutaj ogólnych reguł określających sposób budowy sieciowych modeli dowolnych obiektów. Można jednak sformułować zasady, które ułatwiają wykonanie takiego zadania w sytuacjach typowych. Najważniejsze wydaje się hierarchiczne podejście do problemu. Jeżeli więc mamy przedstawić model wywoływania określonej procedury w pętli nieskończonej, to najbardziej ogólnym modelem takiego „algorytmu" może być sieć z rys. 4. Gdy jednak chcemy wyraźnie rozróżnić wywołanie, wykonywanie i powrót z procedury, to wystarczy wykonać przekształcenie pokazane na rys. 5. Otrzymujemy w ten sposób bardziej szczegółowy model niższego poziomu (rys. 6). Na każdym kolejnym poziomie tworzenia modelu dowolnego algorytmu, pomocne mogą być trzy następne proste reguły modelowania za pomocą sieci Petriego fragmentów algorytmów sekwencyjnych (schematów blokowych). Reguły te są zilustrowane na rys. 7.

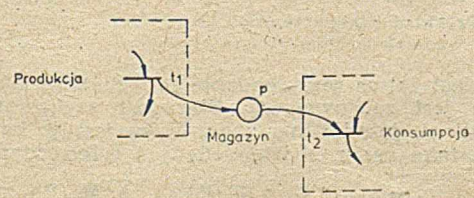


Rys. 6. Wywoływanie procedury w pętli



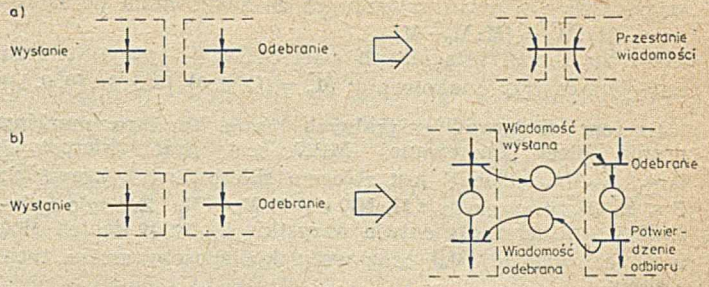
Rys. 7. Przykłady reguł modelowania algorytmów

Najbardziej typowym zastosowaniem sieci Petriego jest jednak modelowanie zjawisk zachodzących asynchronicznie i przedstawienie współdziałania różnych części złożonych systemów. W tym zakresie klasycznym przykładem zastosowania sieci są modele magazynów lub buforów. W modelach tych przyjmuje się, że bufor jest reprezentowany przez miejsce, a liczba znajdujących się w nim znaczników określa bieżącą zajętość bufora. W najprostszej postaci takiego modelu (rys. 8) wzbudzenie t_1 oznacza umieszczenie następnego elementu w magazynie, a wzbudzenie t_2 — pobranie elementu z magazynu. Model z rys. 8 może stanowić pierwszy krok w rozwiązywaniu bogato prezentowanego w literaturze zagadnienia modelowania klasycznego problemu współbieżności **producent-konsument.**



Rys. 8. Model działania magazynu

W prezentacji zjawisk współbieżnych formułuje się zasady tworzenia modelu współdziałania różnych elementów modelowanego obiektu. Najczęściej spotyka się podejście,



Rys. 9. Reguły modelowania przesłania wiadomości

w którym najpierw oddzielnie modeluje się współpracujące elementy, a następnie, stosując odpowiednie reguły, modele te łączy się ze sobą. Rys. 8 stanowi przykład zastosowania jednej z takich reguł, zgodnie z którą za pomocą dodatkowego dzielonego miejsca łączy się ze sobą dwa przejścia reprezentujące wzajemnie zależne operacje, wykonywane przez różne elementy (np. umieszczenie i pobranie elementu w magazynie, nadanie i odebranie wiadomości, operacje V i P na semaforze). Na przykładzie nadania i odebrania wiadomości można pokazać również użyteczność innych reguł — zilustrowanych na rys. 9. Rodzaj stosowanej reguły zależy bowiem od natury modelowanego zjawiska lub pożądanej szczegółowości modelu [7]. Regułę z rys. 9a stosuje się wtedy, gdy zarówno nadajnik jak i odbiornik muszą być przygotowane równocześnie, co warunkuje przeprowadzenie transmisji. Z kolei reguła z rys. 9b odnosi się (podobnie jak rys. 8) do sytuacji, gdy takie wymaganie nie obowiązuje. Reguła z rys. 9b jest też najprostszym modelem transmisji z potwierdzeniem. Jeszcze inne reguły muszą być stosowane, gdy model ma wyrażać fizyczną, sprzętową naturę takich operacji. Propozycje takich reguł można znaleźć w [10].

Analiza własności sieci

Rolę sieci Petriego najlepiej wyraża fakt, że pozwalają one badać istotne własności modelowanych obiektów przed przystąpieniem do konkretnej ich realizacji. Rodzaje badanych własności zależą przede wszystkim od potrzeb użytkownika oraz teoretycznej i praktycznej możliwości wykonania odpowiedniej analizy modelu. Pewne ogólne własności modelu są jednak badane prawie zawsze, ponieważ odzwierciedlają cechy podstawowe, istotne dla różnych rodzajów modelowanych obiektów. Część z tych własności może być sprawdzona już na podstawie samej struktury sieci — bez uwzględniania znakowań [4], lecz większość ma jednak charakter dynamiczny i zależy od sposobu, w jaki w danej sieci mogą zmieniać się znakowania. Badanie dynamicznych własności sieci Petriego na podstawie zmian znakowań jest tylko jedną z możliwych do zastosowania metod. Jest to metoda intuicyjnie bardzo prosta, w zasadzie wspólna dla różnych klas sieci, ma czytelną interpretację w teorii grafów oraz w językach programowania i dlatego — mimo pewnych wad (ograniczone zastosowanie dla niektórych sieci, znaczne koszty implementacji) — zostanie poniżej krótko przedstawiona. Przedtem jednak konieczne jest wprowadzenie kilku pojęć i ustalenie pomocniczej notacji.

Znakowanie M_j jest **bezpośrednio osiągalne** (ang. directly reachable) ze znakowania M_i (ozn. $M_i \rightarrow M_j$) wtedy i tylko wtedy, gdy w znakowaniu M_i istnieje przygotowane przejście t, takie że jego wzbudzenie przekształca znakowanie M_i w M_j. Zbiór przejść przygotowanych przez znakowanie M oznacza się T(M), a zmianę znakowania z M_i na M_j w wyniku wzbudzenia przejścia t zapisuje się $M_i \xrightarrow{t} M_j$.

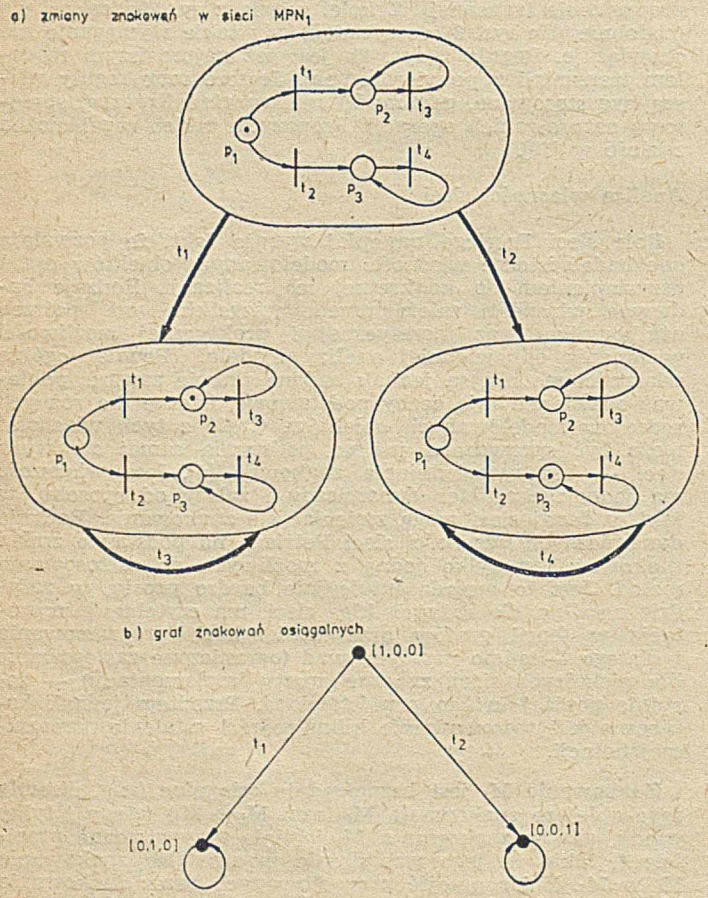
Znakowanie M_j jest **osiągalne** (ang. reachable) ze znakowania M_i, jeżeli istnieje sekwencja (ciąg) znakowań $M_{i_0}, M_{i_1}, \ldots, M_{i_k}$ taka, że $M_{i_0} = M_i$ i $M_{i_k} = M_j$ i $\forall l \leqslant l \leqslant k$: $M_{i_{l-1}} \xrightarrow{t_{il}} M_{i_l}$. Ciąg przejść $\delta = (t_{i_1}, t_{i_2}, \ldots t_{i_k})$ wzbudzanych kolejno nazywa się sekwencją wzbudzeń. Dla zapisania, że znakowanie M_j jest osiągalne z M_i przyjmuje się notację $M_i \xrightarrow{*} M_j$. Jeżeli przejście od M_i do M_j następuje w wyniku sekwencji wzbudzeń δ, używa się notacji $M_i \xrightarrow{\delta} M_j$. Dla wygody przyjmuje się z definicji, że sekwencja może być pusta (ozn. λ), czyli prawdziwy jest zapis $M \xrightarrow{\lambda} M$.

Zbiorem znakowań osiągalnych (ang. forward marking class) M_0 sieci znakowanej $MPN = \langle PN, M_0 \rangle$ jest zbiór

wszystkich znakowań osiągalnych ze znakowania początkowego M_o: $\overrightarrow{M_o} = \{M: M_o \overset{*}{\rightarrow} M\}$.
Zbiór znakowań osiągalnych sieci MPN_1 (z rys. 1) tworzą następujące trzy znakowania: $\overrightarrow{M_o} = \{[1,0,0], [0,1,0], [0,0,1]\}$

Grafem znakowań osiągalnych (ang. forward marking graph) sieci znakowanej MPN jest graf skierowany $G = \langle \overrightarrow{M_o}, F \rangle$, gdzie F jest zbiorem skierowanych krawędzi: $F \subseteq \overrightarrow{M_o} \times \overrightarrow{M_o}$, przy czym $\langle M_i, M_j \rangle \in F \Leftrightarrow M_i \rightarrow M_j$. Zgodnie z definicją, graf G reprezentuje wszystkie stany w jakich sieć może się znaleźć ($\overrightarrow{M_o}$) oraz wszystkie sposoby zmian tych stanów.

Wyznaczenie grafu znakowań osiągalnych dla danej sieci znakowanej jest bardzo proste. Najpierw — wyznacza się zbiór $T(M_0)$ przejść przygotowanych przez znakowanie początkowe. Następnie dla każdego z tych przejść wyznacza się nowe znakowanie, osiągalne bezpośrednio ze znakowania początkowego oraz nowy zbiór przejść przygotowanych. Każdy taki elementarny krok oznacza utworzenie jednej krawędzi grafu znakowań osiągalnych. Ten sam tryb postępowania należy kontynuować dla wszystkich nowo powstałych znakowań. Proces zmian znakowań w sieci oraz tworzenie grafu znakowań osiągalnych dla sieci MPN_1 ilustruje rys. 10a. Na rys. 10b pokazano typowy sposób przedstawiania grafu G. Węzłom są przyporządkowane znakowania, a krawędzie są etykietowane nazwami przejść, których wzbudzenie powoduje odpowiednią zmianę znakowania.



Rys. 10. Konstruowanie grafu znakowań osiągalnych

W praktyce proces tworzenia grafu znakowań osiągalnych jest dla większości konstruowanych sieci Petriego skończony. Koniecznym i wystarczającym warunkiem skończoności grafu znakowań osiągalnych jest to, aby sieć była ograniczona. **Sieć** znakowaną MPN **nazywa się ograniczoną** (ang. bounded), jeżeli:

$$\exists k > 0 \, \forall M (M_o \overset{*}{\rightarrow} M) \wedge \forall p \in P: M(p) \leqslant k$$

Niekiedy sieć ograniczoną, dla której $k = 1$, nazywa się **siecią bezpieczną** (ang. safe).

Konieczność analizy sieci, które nie są ograniczone, występuje raczej rzadko, ponieważ tylko sporadycznie mamy do czynienia z praktyczną interpretacją sytuacji, w której w pewnym miejscu sieci może zachodzić kumulacja nieograniczonej liczby znaczników.

Jeżeli natomiast sieć jest ograniczona, to spełnianie przez daną sieć pozostałych zdefiniowanych poniżej własności można sprawdzać analizując graf osiągalnych znakowań tej sieci.

Sieć znakowana MPN **jest żywa** (ang. live), jeżeli dla każdego znakowania osiągalnego ze znakowania początkowego możliwe jest wzbudzenie (przygotowanie) każdego przejścia sieci przez dalszą sekwencję wzbudzeń, czyli wtedy i tylko wtedy, gdy:

$$\forall M_j (M_o \overset{*}{\rightarrow} M_i) \wedge \forall t \in T \quad \exists M_j: M_i \overset{*}{\rightarrow} M_j \wedge t \in T(M_j)$$

Ważną własnością modelowanych sytemów, które można badać za pomocą sieci Petriego, jest możliwość występowania **zakleszczenia** (ang. deadlock), czyli takiej sytuacji, w której działanie systemu zostaje zawieszone bez możliwości wznowienia.

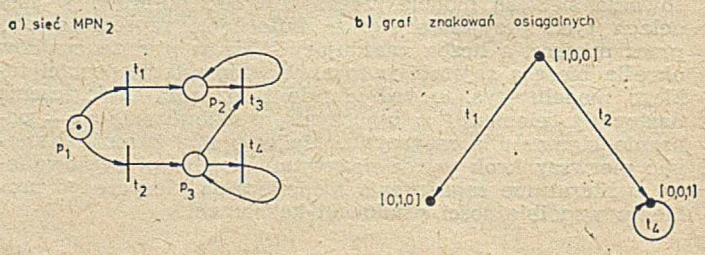
Znakowanie M sieci MPN reprezentuje zakleszczenie, jeżeli zbiór przejść przygotowanych przez to znakowanie jest pusty (tzn. $T(M) = \Phi$), czyli jeżeli w grafie znakowań osiągalnych istnieje węzeł końcowy.

Żywość sieci jest warunkiem wystarczającym, aby nie występowały w niej zakleszczenia. Nie jest jednak warunkiem koniecznym, czego dowodem jest sieć MPN_1, która w myśl podanej definicji (istnieją różne definicje sieci żywej) nie jest żywa, ale też nie występuje w niej zakleszczenie.

Sieć znakowana **jest trwała** (ang. persistent), jeżeli w każdym znakowaniu, osiągalnym ze znakowania początkowego, istnieje co najmniej jedno przygotowane przejście, czyli wtedy i tylko wtedy, gdy:

$$\forall M \, (M_o \overset{*}{\rightarrow} M): T(M) \neq \Phi$$

Badanie trwałości sieci jest równoważne sprawdzeniu, czy w sieci tej nie może wystąpić zakleszczenie (sieć trwała nazywa się nawet czasami **siecią bez zakleszczenia** — ang. deadlock-free). Sprawdzenie tej niezwykle istotnej własności każdego modelowanego systemu jest wykonalne za pomocą analizy grafu znakowań osiągalnych. Sieć MPN_1 jest wprawdzie trwała, wystarczy jednak wprowadzić do niej dodatkową krawędź (rys. 11a) a wzbudzenie przejścia t_1 prowadzi do zakleszczenia, co widać na grafie znakowań osiągalnych (rys. 11b), w którym występuje wierzchołek końcowy. Trwałość sieci nie wyklucza jednak, że może w niej wystąpić **zakleszczenie częściowe** (ang. partial deadlock) lub **lokalne**, które wyraża się przez cykliczne wzbudzenie pewnego podzbioru przejść tej sieci, bez możliwości opuszczenia cyklu. W sieciach żywych zakleszczenie lokalne nie występuje, ale udowodnienie, że sieć jest żywa, a nawet sprawdzenie tego faktu na podstawie analizy jej grafu znakowań osiągalnych jest z reguły bardzo trudne, zwłaszcza dla bardziej rozbudowanych sieci. Badanie żywości sieci na podstawie grafu znakowań polega bowiem na sprawdzeniu, czy w grafie tym istnieje cykl obejmujący wszystkie węzły (znakowania) i czy każdemu przejściu sieci odpowiada krawędź należąca do tego cyklu.



Rys. 11. Przykład zakleszczenia

Natomiast łatwe jest zbadanie innych własności, jak np. czy sieć jest zachowawcza. **Sieć jest zachowawcza** (ang conservative), jeżeli suma znaczników jest w każdym znakowaniu taka sama, a więc wtedy i tylko wtedy, gdy:

$$\forall M (M_o) \overset{*}{\rightarrow} M: \sum_{1 \leqslant i \leqslant \mathrm{card}(P)} M(P_i) = \sum_{1 \leqslant j \leqslant \mathrm{card}(P)} M_o(P_j)$$

Zachowawczość jest szczególnie ważną cechą takich sieci, w których — ze względu na modelowane obiekty — znacznikom przypisuje się istotną interpretację ilościową. Graf znakowań osiągalnych umożliwia także łatwe sprawdzenie,

czy w sieci występują konflikty. Sieć znakowaną nazywa się **siecią bez konfliktów** (ang. conflict-free), jeżeli w każdym znakowaniu osiągalnym ze znakowania początkowego nie występuje taka para przygotowanych przejść, która ma chociaż jedno wspólne miejsce wejściowe:

$$\forall M(M_0 \xrightarrow{*} M): \{t \in T(M): (\exists t_i \in T(M): t \neq t_i \wedge {}^{\bullet}t \cap {}^{\bullet}t_i \neq \Phi)\} = \Phi$$

Podana definicja konfliktu w sieci oznacza, że wzbudzenie jednego z konfliktowych przygotowanych przejść może (ale nie musi — np. gdy liczba znaczników w „konfliktowym" miejscu jest większa od 1) spowodować wycofanie przygotowania drugiego konfliktowego przejścia. Z sytuacją taką mamy do czynienia np. w początkowym znakowaniu sieci MPN_1. W sieciach bez konfliktów wzbudzenie dowolnego przejścia nie pozbawia przygotowania żadnego z innych przygotowanych przejść. Dla obiektów modelowanych takimi sieciami może to oznaczać, że wykonanie jednej czynności nie blokuje możliwości wykonania przez ten obiekt (lub w tym obiekcie) innej czynności dozwolonej w danym stanie operacji. Obie operacje mogą więc być wykonywane równolegle. Jedyną możliwością utraty przygotowania przez dane przejście jest wtedy jego wzbudzenie.

Istnieją również inne definicje konfliktu w sieci. W niektórych definicjach warunek bezkonfliktowości przejść nakłada się na wszystkie przejścia sieci, a nie tylko na przejścia przygotowane przez dane znakowanie. Tak zdefiniowana własność może być badana na podstawie samej struktury sieci, bez konieczności tworzenia grafu znakowań osiągalnych.

Przedstawiony powyżej zestaw własności nie wyczerpuje oczywiście listy cech, których badanie umożliwiają sieci Petriego. Konkretne pytania, dotyczące modelowania obiektu, zależą bowiem od sposobu opisu obiektu oraz interpretacji nałożonej na poszczególne elementy modelującej go sieci. W zakresie zjawisk współbieżnych przykładowe pytanie może dotyczyć spełnienia warunku wykluczania wzajemnego. Zależnie od tego, czy wykluczanie ma dotyczyć stanów czy czynności, oraz zależnie od przyjętej interpretacji modelu, sprawdzenie tego warunku można sprowadzić do zbadania, czy istnieje znakowanie, w którym w kilku konkretnych miejscach występują znaczniki, lub w którym jest przygotowanych kilka konkretnych przejść. Wystąpienie takiego znakowania oznaczałoby bowiem naruszenie badanego warunku przez modelowany obiekt. Inne własności, które można badać za pomocą analizy grafu znakowań osiągalnych mogą dotyczyć na przykład:

- istnienia cykli w grafie
- istnienia konkretnej sekwencji wzbudzeń w danym cyklu lub w całym grafie
- relacji (poprzedzenie, następstwo, równoległość) między sekwencjami wzbudzeń
- relacji między liczbami wzbudzeń konkretnych przejść
- występowania określonych niezmienników (np. stałych zależności między obecnością znaczników w pewnych miejscach a przygotowaniem konkretnych przejść). Jedyne ograniczenie w badaniu takich skomplikowanych lub bardzo szczegółowych własności stanowią praktyczne możliwości analizy.

LITERATURA

[1] Agerwala T., Flynn M. J.: Comments on capabilities, limitations and „correctness" of Petri nets. Proc. of the 1st Annual Symposium on Computer Architecture, Gainesville, USA, pp. 81—86, 1973

[2] Andre C.: The behaviour of a Petri net on a subset of transitions. RAIRO Automatique/Systems Analysis and Control, Vol. 17, No. 1, pp. 5—21, 1983

[3] Ayache J. M., Azema P., Diaz M.: Observer — a concept for on-line detection of control errors in concurrent systems. Proc. of FTCS-9, Madison, pp. 79—86, 1979

[4] Berthomieu B.: Analyse structurelle des reseaux de Petri. Methodes et outils, these de docteur ing., Univ. de Toulouse, 1979

[5] Brams G. W.: Resaux de Petri: theorie et practique. Masson, Paris, 1983

[6] Brauer W. (ed.): Net theory and applications. Proc. of Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979 (także w: Lecture Notes in Computer Science, No 84, Springer Verlag, 1980)

[7] Diaz M.: Modelling and analysis of communication and cooperation protocols using Petri nets based models. W: Sunshine C. (ed.): Protocol Specification, Testing and Verification, North-Holland Publ. Co., pp. 465—510, 1982

[8] Genrich H. J., Lautenbach K.: System modelling with high-level Petri nets. Theoretical Computer Science, Vol. 13, pp. 109—136, 1981

[9] Genrich H. J., Stankiewicz-Wiechno E.: A dictionary of some basic notions of net theory. W. [6]

[10] Gondzio M., Budkowski S.: Modelling and verifying asynchronous cooperation between microprogrammed units. Proc. of the 13th Fault-Tolerant Computing Symposium, Milano, pp. 74—77, 1983

[11] Iszkowski W., Maniecki M.: Programowanie współbieżne. WNT, Warszawa, 1982

[12] Keller R. M.: Generalized Petri nets as models for system verification. Proc. of the Conference on Petri Nets and Related Methods, Springer Lecture Notes

[13] Moella M., Pulou J., Sifakis J.: Resaux de Petri synchronises. Res. Rep. 80, Univ. Scientifique et Medical de Grenoble, 1977

[14] Peterson J. L.: Petri nets theory and the modelling of systems. Prentice Hall, 1981

[15] Petri C. A.: Fundamentals of a theory of asynchronous information flow. Proc. of the IFIP Congress, Amsterdam, pp. 386—390, 1962

[16] Petri C. A.: Introduction to general net theory. W: [6]

[17] Ramchandani C.: Analysis of asynchronous concurrent systems by timed Petri nets. MAC-TR 120, Project MAC-MIT, 1974

[18] Reisig W.: Petrinetze, eine Einfuhrung. Springer Verlag, 1982

[19] Zuberek W.: Analiza efektywności jednostek centralnych maszyn cyfrowych oraz niektóre metody jej zwiększenia. Prace IINTE, nr 26, 1979

[20] Jensen K., Huber P., Larsen M. N., Martinsen I. M.: Petri Net Package User's Manual. DAIMI MD-46, Aarhus University, 1983.

KONFERENCJE

Convention Informatique '87

Już po raz osiemnasty w paryskim Pałacu Kongresów w dniach od 6 do 10 kwietnia 1987 odbędzie się doroczna, międzynarodowa konferencja Convention Informatique, związana z wystawą SICOB. Wystawa ta, odbywająca się tradycyjnie we wrześniu, w ostatnich latach jest organizowana w dwóch terminach: wiosennym i jesiennym. W bieżącym roku po raz pierwszy Convention Informatique odbędzie się właśnie w terminie wiosennym. Organizatorzy podkreślają, że jest to w chwili obecnej największa w Europie impreza informatyczna, argumentując liczbą uczestników sięgającą już ok. 3500 osób (w tym ok. 1/4 z zagranicy). Językami oficjalnymi konferencji, podobnie jak w poprzednich latach, będą angielski i francuski z zapewnieniem bieżącego tłumaczenia. Należy wspomnieć, że w bieżącym roku następuje również zmiana lokalizacji wystawy SICOB, która odbędzie się nie w tradycyjnej, urosłej do symbolu tej imprezy, hali CNIT w dzielnicy La Defence, lecz na terenach wystawowych Parc d'Expositions Paris-Nord.

Głównymi tematami konferencji, które stanowić będą wyodrębnione sesje obrad, są:

- **rynki i trendy,**
- **rozwój technologiczny,**
- **zastosowania,**
- **mikrokomputery,**
- **zarządzanie i aspekty ekonomiczne.**

Dalsze szczegóły uzyskać można od sekretariatu konferencji Informatique 4, place de Valois 75001 Paris (France), tel (1) 42 61 46 21 oraz (1) 42 61 52 42.

Równocześnie z obradami Convention Informatique, i w tym samym Pałacu Kongresów, odbywać się będzie w dniach od 6 do 9 kwietnia międzynarodowa konferencja Convention Automatique Productique 1987 poświęcona narzędziom projektowania, eksploatacji, nadzoru i konserwacji systemów automatyzacji produkcji. Tematyka tej konferencji przypomina profil konferencji międzynarodowych organizowanych przez IFAC (Międzynarodowa Federacja Automatycznego Sterowania).

Szczegółowe informacje można uzyskać w sekretariacie konferencji: Secretariat de la Convention Automatique Productique 1987, 11, rue Hamelin, F-75783 Paris Cedex 16 (France), tel. 33 (1) 45 05 14 27.

WOJCIECH CELLARY
Instytut Automatyki
Politechnika Poznańska

POLSKIE TOWARZYSTWO INFORMATYCZNE

Systemy wielomikroprocesorowe (2)

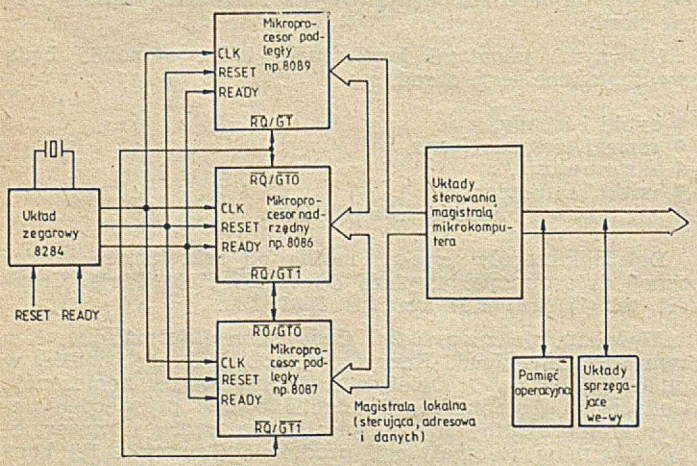
W drugiej części artykułu omówiono systemy z dzielonymi magistralami — lokalną i systemową.

Systemy wielomikroprocesorowe z dzieloną magistralą lokalną

Architektonicznie najbliższe omówionym systemom jednomikroprocesorowym z kanałem DMA są systemy wielomikroprocesorowe z dzieloną magistralą lokalną. Magistrala lokalna jest utworzona przez linie bezpośrednio dochodzące do końcówek mikroprocesora.

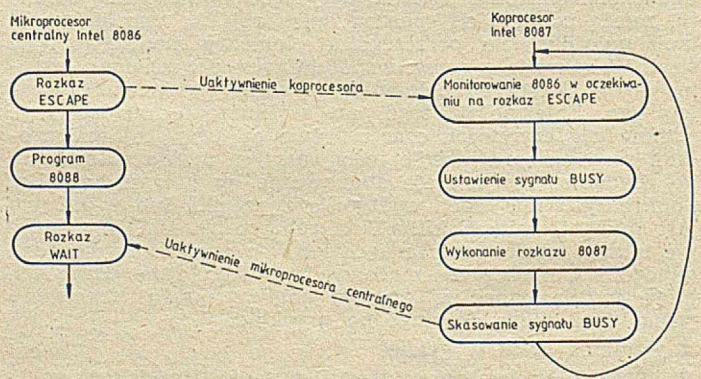
Możliwość budowy systemów wielomikroprocesorowych z dzieloną magistralą lokalną wynika z wbudowania do mikroprocesora Intel 8086 i innych mikroprocesorów tej rodziny pewnego systemu arbitrażu tej magistrali. Umożliwia on współpracę trzech mikroprocesorów. Jeden z nich pełni rolę nadrzędną i nadzoruje dostęp do magistrali dwóch pozostałych, z których jeden ma wyższy priorytet niż drugi. Wymiana sygnałów żądania przydziału magistrali lokalnej, potwierdzenia przyjęcia żądania i zwolnienia magistrali między mikroprocesorem nadrzędnym a jednym z mikroprocesorów podległych odbywa się na jednej linii sygnałowej oznaczonej $\overline{RQ/GT}$ (ang. request/grant). Żądanie przydziału magistrali lokalnej ze strony mikroprocesora podległego jest zgłaszane do mikroprocesora nadrzędnego przez podanie impulsu na linię $\overline{RQ/GT}$. Mikroprocesor nadrzędny, po otrzymaniu tego żądania i po zakończeniu wykonywania bieżącego cyklu maszynowego, wysyła tą samą linią impuls potwierdzenia przyjęcia żądania i odłącza się od magistrali. Oczywiście może on kontynuować działanie pobierając rozkazy ze swojej wewnętrznej kolejki rozkazów. Mikroprocesor podległy po zrealizowaniu dostępu do magistrali odłącza się od niej i informuje o tym fakcie mikroprocesor nadrzędny przesyłając impuls linią $\overline{RQ/GT}$.

Architekturę systemu wielomikroprocesorowego z dzieloną magistralą lokalną przedstawiono na rys. 1. Ten rodzaj organizacji systemu wielomikroprocesorowego wymaga synchronizacji wszystkich mikroprocesorów, dlatego w systemie znajduje się jeden wspólny układ zegarowy 8284.



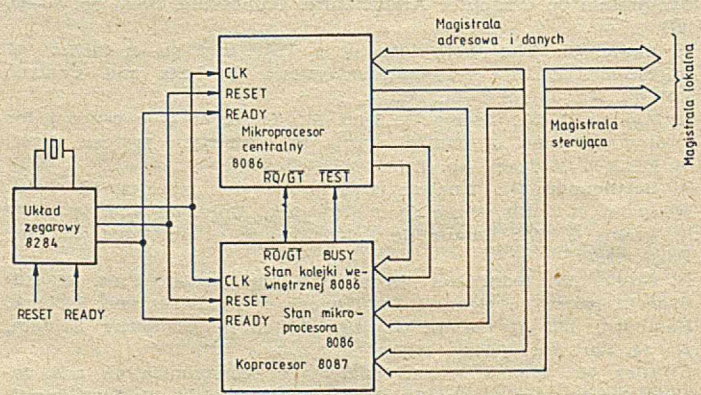
Rys. 1. Architektura systemu wielomikroprocesorowego z dzieloną magistralą lokalną

Istotną zaletą systemów wielomikroprocesorowych z dzieloną magistralą lokalną jest to, że nie wymagają one stosowania specjalnych dodatkowych układów. Odpowiednie mechanizmy są wbudowane w same mikroprocesory, choć należy zwrócić uwagę, że mechanizmy te nie są wystarczające do połączenia w systemie kilku mikroprocesorów centralnych Intel 8086. Głównym przeznaczeniem omawianej architektury jest łączenie mikroprocesorów specjalizowanych. Przykładową konfiguracją tego rodzaju jest połączenie mikroprocesora centralnego Intel 8086 (lub Intel 8088) ze specjalizowanym mikroprocesorem we-wy Intel 8089. Ten drugi łączy w sobie cechy mikroprocesorów centralnych (czyli zdolność do wykonywania programów) z cechami kanałów DMA, a więc z możliwością szybkiej, ciągłej transmisji danych. Schemat synchronizacji na poziomie zadań wykonywanych przez dwa mikroprocesory przedstawiono na rys. 2. Jak widać, synchronizacja ta odbywa się z wykorzystaniem operacji we-wy, wspólnej pamięci operacyjnej i przerwań.



Rys. 2. Synchronizacja mikroprocesorów: centralnego i we-wy na poziomie zadań

Inną typową konfiguracją systemu wielomikroprocesorowego z dzieloną magistralą lokalną jest połączenie mikroprocesora centralnego Intel 8086 (lub Intel 8088) z koprocesorem, np. koprocesorem arytmetycznym Intel 8087. Koprocesor nie wykonuje własnego programu i z tego względu zawsze towarzyszy innej jednostce centralnej. Jego zadaniem jest rozszerzenie listy rozkazów mikroprocesora centralnego i wykonywanie tych uzupełniających rozkazów równolegle z programem mikroprocesora centralnego. Tak więc koprocesor Intel 8087 rozszerza listę rozkazów mikroprocesora Intel 8086 o złożone operacje arytmetyczne, w szczególności zmiennoprzecinkowe.

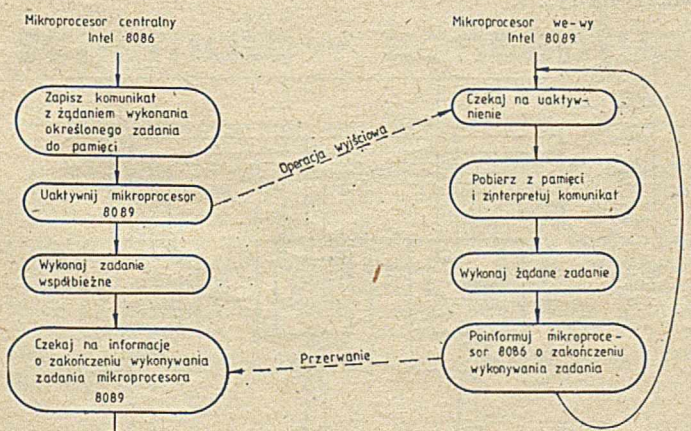


Rys. 3. System wielomikroprocesorowy z mikroprocesorem centralnym i koprocesorem

Architekturę systemu wielomikroprocesorowego z dzieloną magistralą lokalną (w konfiguracji: mikroprocesor centralny — koprocesor) przedstawiono na rys. 3. Współpraca koprocesora z mikroprocesorem centralnym polega na monitorowaniu rozkazów pobieranych przez mikroprocesor centralny Intel 8086 w oczekiwaniu na rozkazy przeznaczone dla koprocesora. Są to 64 rozkazy ESCAPE. Koprocesor rozpoznaje rozkazy ESCAPE (w szczególności odróżnia je od danych) na podstawie badania stanu mikroprocesora centralnego. Ponadto koprocesor bada stan kolejki wewnętrznej mikroprocesora centralnego (z faktu pobrania rozkazu przez mikroprocesor centralny nie wynika bowiem jeszcze jego wykonanie).

Mikroprocesor centralny ignoruje rozkazy ESCAPE. Wykonanie takiego rozkazu odpowiada wykonaniu klasycznego rozkazu „nic nie rób", z tą różnicą, że jeśli tryb adresowania rozkazu ESCAPE wskazuje na istnienie argumentu w pamięci operacyjnej, to argument ten jest pobierany. Koprocesor ma analogiczną kolejkę wewnętrzną jak mikroprocesor centralny i równolegle do mikroprocesora centralnego pobiera od niej te same rozkazy. W wypadku stwierdzenia (na podstawie kodu rozkazu i stanu kolejki wewnętrznej mikroprocesora centralnego) przeznaczonego dla niego rozkazu ESCAPE, rozpoczyna jego równoległe wykonanie. Określone pola rozkazu ESCAPE, ignorowane przez mikroprocesor centralny (łącznie pięć bitów), są traktowane przez koprocesor jako kod rozkazu z jego listy rozkazów. Jeśli tryb adresowania rozkazu ESCAPE wskazuje na umieszczenie argumentu rozkazu w rejestrze, to zawartość trzybitowego pola adresowania rejestru jest interpretowana przez koprocesor jako rozszerzenie kodu rozkazu (koprocesor nie ma oczywiście dostępu do rejestrów mikroprocesora centralnego). Jeśli tryb adresowania rozkazu ESCAPE wskazuje na umieszczenie argumentu rozkazu w pamięci operacyjnej, to mikroprocesor centralny pobiera ten argument, a koprocesor przechwytuje go łącznie z adresem. Gdy w interpetacji koprocesora pobrana dana jest rzeczywiście argumentem rozkazu, to wykorzystuje on ją bezpośrednio. Jeśli dana jest częścią argumentu, to na podstawie zapamiętanego adresu koprocesor pobiera dalsze bajty argumentu, uzyskując dostęp do magistrali lokalnej przez wysłanie odpowiednich sygnałów na linię $\overline{RQ/GT}$. Wreszcie, jeśli dana pobrana przez mikroprocesor centralny ma być dla koprocesora wynikiem operacji, to koprocesor ignoruje ją, zapamiętując jedynie jej adres, pod który prześle wynik wykonanego rozkazu, uzyskując dostęp do magistrali lokalnej jak poprzednio.

Przebieg synchronizacji między mikroprocesorem centralnym a koprocesorem na poziomie rozkazów, odpowiadający poziomowi zadań w wypadku mikroprocesora centralnego i mikroprocesora we-wy, przedstawiono na rys. 4.



Rys. 4. Synchronizacja mikroprocesora centralnego i koprocesora na poziomie rozkazów

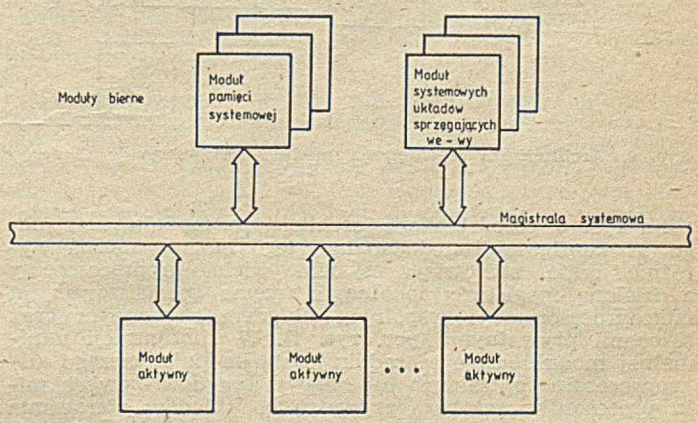
Potrzeba synchronizacji zachodzi również w momencie, gdy mikroprocesor centralny musi skorzystać z wyników równolegle wykonywanej przez koprocesor operacji, a nie ma pewności czy operacja ta już się zakończyła. Do tego rodzaju synchronizacji służy rozkaz WAIT wykonywany przez mikroprocesor centralny. Działanie tego rozkazu jest uzależnione od stanu wejścia $\overline{TEST}$ mikroprocesora, na które podaje się normalnie sygnał wyjściowy BUSY koprocesora. Sygnał ten ma wartość logiczną 1 od momentu zdekodowania rozkazu ESCAPE przez koprocesor do momentu zakończenia wykonywania rozkazu. Jeśli sygnał $\overline{TEST}$ przyjmuje wartość 0, to działanie rozkazu WAIT jest równoważne działaniu rozkazu „nic nie rób". Jeśli natomiast sygnał $\overline{TEST}$ przyjmuje wartość 1, to rozkaz WAIT powoduje zawieszenie działania mikroprocesora centralnego aż do zmiany stanu tego sygnału.

Powracając do problemów architektury systemów wielomikroprocesorowych z dzieloną magistralą lokalną należy stwierdzić, że korzystną i zalecaną konfiguracją jest współdzielenie magistrali lokalnej przez trzy mikroprocesory: centralny 8086, we-wy 8089 i koprocesor 8087 (por. rys. 3). Wówczas, ze względu na to, że mikroprocesor we-wy prowadzi szybkie transmisje danych uwarunkowane czasowo, nadaje mu się wyższy priorytet dołączając do linii $\overline{RQ/GT}$ mikroprocesora centralnego. W wypadku jednoczesnego zgłoszenia żądania przydziału magistrali lokalnej przez mikroprocesor we-wy i koprocesor, w pierwszej kolejności zostanie spełnione żądanie mikroprocesora we-wy. Jeżeli jednak magistrala lokalna jest przydzielona koprocesorowi w momencie zgłoszenia żądania przez mikroprocesor we-wy, to musi on czekać na jej zwolnienie. Aby zminimalizować to oczekiwanie, linię $\overline{RQ/GT}$ mikroprocesora we-wy łączy się dodatkowo z linią $\overline{RQ/GT1}$ koprocesora (por. rys. 3). Wówczas, na mocy opisanego uprzednio mechanizmu działania sygnałów $\overline{RQ/GT}$ w mikroprocesorze centralnym, koprocesor zwalnia magistralę po otrzymaniu żądania od mikroprocesora we-wy natychmiast po zakończeniu bieżącego cyklu maszynowego, nawet jeśli potrzebuje dalszych cykli do dokończenia wykonywania swego rozkazu.

Inną zalecaną konfiguracją mikroprocesorów w omawianej architekturze jest połączenie dwóch mikroprocesorów we-wy w celu zwiększenia liczby kanałów transmisyjnych. W tym wypadku jeden z mikroprocesorów programuje się jako nadrzędny, a drugi jako podległy. Pozostałe mechanizmy ich wzajemnej współpracy pozostają bez zmian.

Systemy wielomikroprocesorowe z dzieloną magistralą systemową

Podstawową zaletą architektury systemów wielomikroprocesorowych z dzieloną magistralą lokalną jest możliwość budowy takich systemów bez specjalnych układów pomocniczych, przy wykorzystaniu mechanizmów wbudowanych w mikroprocesory. Architekturę tę cechują jednak pewne istotne ograniczenia. Przede wszystkim jest ona przewidziana do łączenia mikroprocesorów specjalizowanych. Wykorzystując wyłącznie mechanizmy wbudowane w mikroprocesory rodziny Intel 8086 nie można połączyć w takim systemie kilku mikroprocesorów centralnych. Oczywiście, dobudowując odpowiednią logikę na zewnątrz mikroprocesorów, można skonstruować system wielomikroprocesorowy z dzieloną magistralą lokalną, w którym połączonych byłoby kilka mikroprocesorów centralnych. System taki byłby jednak mało efektywny, ponieważ każdy dostęp do sprzętowych zasobów systemu — pamięci operacyjnej i układów sprzęgających we-wy (por. rys. 1), które są wspólne — byłby potencjalnie konfliktowy. Drogą do zminimalizowania liczby konfliktów i tym samym do zwiększenia efektywności systemu jest rozdział zasobów systemu na **prywatne** (dedykowane) i **systemowe** (współdzielone). Do zasobów prywatnych ma dostęp wyłącznie jeden mikroprocesor centralny systemu, zaś do zasobów systemowych — wszystkie mikroprocesory. Oczywiście rozdział zasobów systemu na prywatne i systemowe wymaga rozdziału magi-



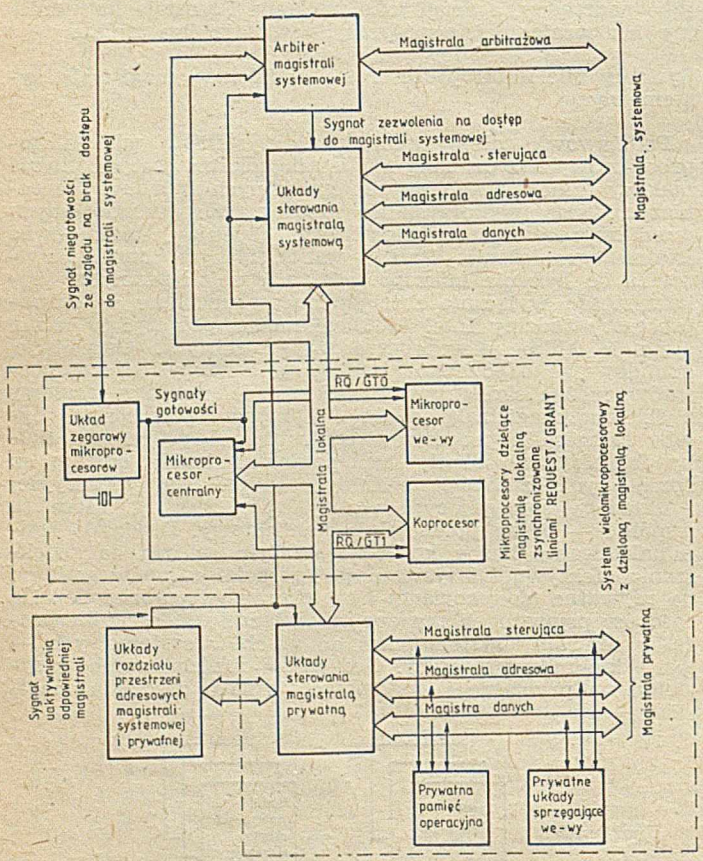
Rys. 5. Architektura systemu wielomikroprocesorowego z dzieloną magistralą systemową

strali lokalnej odpowiednio na magistralę prywatną i systemową. Rozdział ten jest wykonywany w przestrzeni adresowej mikroprocesorów. Po tym jedynie dostęp do zasobów systemowych przez magistralę systemową jest potencjalnie konfliktowy.

Podział zasobów na prywatne i systemowe jest naturalny dla systemów wielomikroprocesorowych. Poszczególne mikroprocesory wykonują bowiem zazwyczaj autonomiczne, równoległe zadania wykorzystując swoje zasoby prywatne i stosunkowo rzadko komunikują się między sobą, co wymaga wykorzystania zasobów systemowych.

Systemy wielomikroprocesorowe o rozdzielonej w przestrzeni adresowej magistrali lokalnej w sposób opisany powyżej nazywają się systemami z **dzieloną** (w czasie) **magistralą systemową.** Ogólną architekturę takiego systemu przedstawiono na rys. 5. W systemie takim wyróżnia się moduły **aktywne** i **bierne,** dołączone do magistrali systemowej. Moduły aktywne, w przeciwieństwie do modułów biernych, mogą samodzielnie ubiegać się o dostęp do magistrali systemowej. Należy podkreślić, że wyróżnikiem modułów aktywnych jest ich zdolność do ubiegania się o dostęp do magistrali systemowej, a nie fakt wyposażenia w mikroprocesor, który skądinąd jest konieczny do uzyskania tej zdolności. Moduły bierne mogą być również wyposażone w mikroprocesory, co wcale nie przesądza o możliwości ich samodzielnego dostępu do magistrali.

Architektura logiczna systemu wielomikroprocesorowego z dzieloną magistralą systemową, przedstawiona na rys. 5, odpowiada na ogół architekturze fizycznej takiego systemu. Oznacza to, że każdy moduł systemu jest zawarty zazwyczaj na jednej płycie.



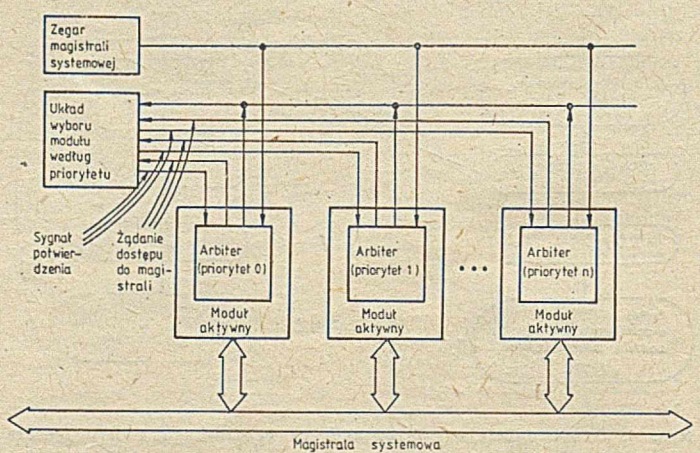
Rys. 6. Architektura wewnętrzna modułu aktywnego

Na rys. 6 przedstawiono architekturę modułu aktywnego. W konfiguracji maksymalnej tego modułu można wyróżnić część stanowiącą system wielomikroprocesorowy z dzieloną magistralą lokalną, z trzema specjalizowanymi mikroprocesorami i wspólnym układem zegarowym oraz zasobami prywatnymi (por. rys. 1). W konfiguracji minimalnej modułu część ta redukuje się do pojedynczego mikroprocesora centralnego lub we-wy, bez dodatkowych mikroprocesorów, zasobów prywatnych, układów sterowania magistralą prywatną i w konsekwencji — układów rozdziału przestrzeni adresowej magistrali systemowej i prywatnej. Oczywiście są możliwe konfiguracje pośrednie. W stosunku do systemu z dzieloną magistralą lokalną moduł aktywny zawiera pewne układy dodatkowe, przede wszystkim układy sterowania magistralą systemową. Z dokładnością do szczegółów układy te nie różnią się od odpowiednich układów sterowania magistralą prywatną.

Rozdział przestrzeni adresowej mikroprocesora między magistralę prywatną i systemową wykonuje układ dekoderów adresów. Układ ten jest dołączony do magistrali adresowej magistrali prywatnej, na tej bowiem magistrali mogą pojawiać sią wszystkie adresy przesyłane przez magistralę lokalną modułu. Po stwierdzeniu, do której przestrzeni — prywatnej czy systemowej — należy dany adres, są uaktywniane układy sterowania odpowiednią magistralą. W wypadku magistrali prywatnej jest to równoznaczne z wykonaniem przesłania danej. Natomiast w wypadku magistrali systemowej jest to dopiero jeden z dwóch warunków koniecznych, których jednoczesne spełnienie jest wymagane dla czynników przesłania danej. Tym drugim warunkiem jest brak zajętości magistrali systemowej przez inny moduł aktywny. Układem odpowiedzialnym za uzyskanie wyłącznego dostępu do magistrali systemowej jest tzw. **arbiter.** W wypadku pojawienia się adresu z systemowej przestrzeni adresowej i stwierdzenia przez arbiter braku zajętości magistrali systemowej, zezwala on na dostęp do niej i jest wykonywane przesłanie danej. W przeciwnym razie arbiter wysyła sygnał niegotowości do układu zegarowego mikroprocesorów modułu, co powoduje odpowiednie wydłużenie cyklu maszynowego, aż do uzyskania dostępu do magistrali.

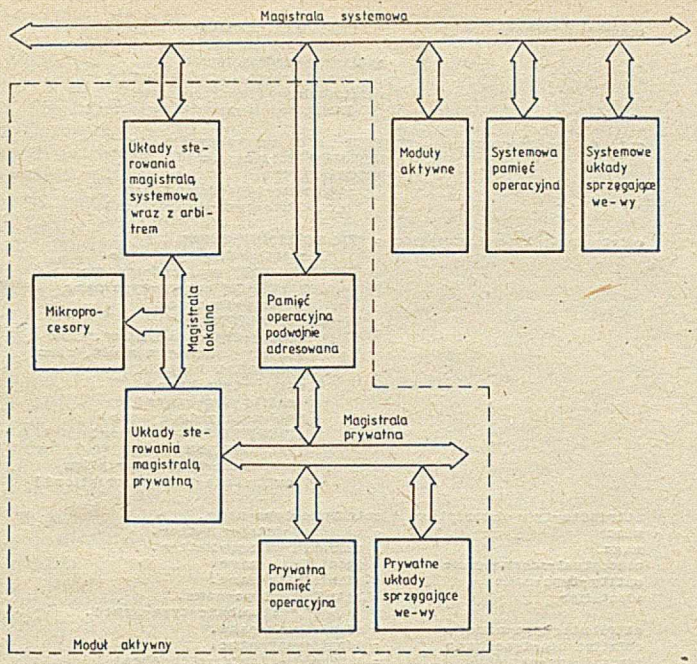
Warto przedstawić pokrótce problem arbitrażu magistrali systemowej, czyli problem konfliktów powstających podczas ubiegania się o dostęp do magistrali systemowej ze strony wielu modułów aktywnych. Najprostszym rozwiązaniem tego problemu byłoby wprowadzenie wspólnego zegara dla wszystkich mikroprocesorów systemu, tak aby zapewnić ich synchroniczną pracę i realizować dostęp do magistrali systemowej cyklicznie. Rozwiązanie takie spowodowałoby bardzo nieefektywną pracę systemu — często magistrala byłaby przydzielona mikroprocesorowi nie żądającemu dostępu do niej, podczas gdy inne oczekiwałyby na dostęp. Z tego względu w stosowanym rozwiązaniu mikroprocesory różnych modułów aktywnych nie są zsynchronizowane między sobą i mogą wysyłać żądania dostępu w dowolnych chwilach. Żądania te są następnie synchronizowane przez arbitry z zegarem magistrali systemowej. Po zsynchronizowaniu jest możliwe określenie, który z ubiegających się o dostęp do magistrali systemowej modułów aktywnych uzyska go. Wybór odbywa się według algorytmu priorytetowego.

Miejsce arbitrów w architekturze systemu wielomikroprocesorowego z dzieloną magistralą systemową przedstawiono na rys. 7.



Rys. 7. Układ arbitrażu magistrali systemowej

Mówiąc o problemie arbitrażu należy wspomnieć o pewnej koniecznej właściwości arbitra magistrali systemowej, a mianowicie zdolności do zapewnienia w niektórych wypadkach ciągłego dostępu do tej magistrali w czasie pełnego cyklu rozkazowego złożonego z kilku cykli maszynowych. Konieczność zagwarantowania takiego dostępu występuje na przykład przy wykonywaniu operacji na semaforach służących do synchronizacji zadań na poziomie systemu operacyjnego. Gdyby nie było możliwości nieprzerwanego dostępu do magistrali systemowej w ciągu pełne-



Rys. 8. System wielomikroprocesorowy z pamięcią operacyjną podwójnie adresowaną

go rozkazu, to mogłoby dojść do sytuacji, w której rozkazy testowania wartości semafora i jego uaktualnienia, wykonywane przez dwa różne mikroprocesory, przeplatałyby się na poziomie cykli maszynowych powodując błędną interpretację wartości semafora. W konsekwencji niemożliwa byłaby synchronizacja zadań w systemie operacyjnym systemu wielomikroprocesorowego z dzieloną magistralą systemową.

Na zakończenie warto powrócić jeszcze do problemu podziału zasobów. Na początku tej części artykułu wyróżniono zasoby prywatne i systemowe. W niektórych rozwiązaniach pojawiają się zasoby będące jednocześnie zasobami prywatnymi i systemowymi. Dotyczy to tzw. **pamięci podwójnie adresowanych** (ang. dual port RAM). Architekturę systemu z taką pamięcią przedstawiono na rys. 8.

Do pamięci podwójnie adresowanej jest możliwy dostęp zarówno od strony magistrali prywatnej modułu, w skład którego wchodzi pamięć, jak również od strony magistrali systemowej. Jest to więc pamięć systemowa, gdyż mają do niej dostęp wszystkie moduły aktywne systemu, a jednocześnie jest to pamięć prywatna danego modułu, o możliwości szybkiego dostępu z pominięciem arbitrażu magistrali systemowej.

Sterowanie pamięci podwójnie adresowanych redukuje liczbę konfliktów w systemie i tym samym wpływa na zwiększenie jego efektywności. Stosowanie tych pamięci jest szczególnie korzystne przy wymianie dużych bloków danych między modułami aktywnymi systemu.

* * *

W artykule naszkicowano główne problemy architektury systemów wielomikroprocesorowych trzeciej generacji. Szerokie upowszechnienie zastosowań tych systemów, również w Polsce, jest zdaniem autora kwestią przewidywanej przyszłości. Upowszechnienie to będzie zależało w dużej mierze od rozwoju systemów operacyjnych systemów wielomikroprocesorowych. Jest to zagadnienie bardzo aktualne i ciekawe, wykraczające jednak poza ramy tego artykułu.

STANISŁAW WRYCZA
Uniwersytet Gdański

Podejścia do tworzenia schematów konceptualnych

Systematyka modeli konceptualnych jest przedmiotem licznych opracowań. Temat ten podejmowali m.in. Biller i Neuhold [3], Kerschberg, Klug i Tsichritzis [10], Olive [14] oraz Kung [11]. Istota przeprowadzonych analiz porównawczych modeli konceptualnych sprowadza się do opracowania tablic krzyżowych, w których podstawowymi parametrami są kategorie pojęciowe, poziomy uogólnienia w procesie modelowania, podstawy matematyczne oraz stopień realizacji pożądanych cech i ról modeli oraz schematów konceptualnych.

Dr hab. STANISŁAW WRYCZA ukończył studia w 1973 r. W 1977 r. doktoryzował się z zakresu skomputeryzowanych systemów sterowania zapasami towarowymi (nagroda III stopnia Ministra NSzWiT). Jego praca habilitacyjna dotyczyła konceptualnego modelowania baz danych organizacji gospodarczych. Odbył staże naukowe m.in. w Wielkiej Brytanii i USA. Jest wiceprzewodniczącym Sekcji Baz Danych oraz przewodniczącym Komisji Bibliotecznej Polskiego Towarzystwa Informatycznego.

Z dokonanych porównań podejść do modelowania konceptualnego wynika, że większość z nich jest ukierunkowana na modelowanie statyki dziedziny przedmiotowej (wycinka świata rzeczywistego) za pomocą trzech kategorii: obiektu (typu obiektu), atrybutu i relacji (typu relacji). **Obiekt** jest jednoznacznie identyfikowanym składnikiem rozpatrywanej dziedziny przedmiotowej, **relacja** — związkiem między dwoma lub kilkoma obiektami, natomiast **atrybut** — charakterystyką obiektu lub relacji.

Większość podejść pomija lub traktuje marginalnie zagadnienia opisu dynamiki dziedziny przedmiotowej. Dynamikę dziedziny przedmiotowej opisuje się za pomocą funkcji, zdarzeń i operacji. **Funkcja** oznacza czynność, wykonywanie czegoś, bądź zakres działania danej organizacji jako całości albo jakiejś jej części [15]. **Zdarzenie** jest bodźcem oddziałującym na dziedzinę przedmiotową oraz inicjującym jedną lub więcej funkcji (lub operacji). **Operacja** jest odpowiednikiem elementarnej czynności tworzenia, skreślenia, modyfikacji lub wyszukiwania, wykonywanej na obiektach, relacjach i atrybutach schematu konceptualnego.

Podobnie marginesowo jak zagadnienia opisu dynamiki dziedziny przedmiotowej jest traktowana kwestia specyfikowania więzów integralności. Zachowanie integralności bazy danych oznacza przechowywanie w niej jedynie poprawnych danych. **Więzy integralności** są to reguły dotyczące poszczególnych składników schematu konceptualne-

go, odzwierciedlające logiczne związki i ograniczania między elementami dziedziny przedmiotowej [12]. Więzy integralności można sklasyfikować ze względu na takie kryteria, jak:

— pojęcia, których dotyczą (obiekty, relacje, atrybuty i wartości)

— czas oddziaływania (odroczone, natychmiastowe)

— dynamika.

Według niektórych ocen 80% zawartości schematu konceptualnego winien zajmować opis więzów integralności. Należy przy tym zaznaczyć, że jedynie część reguł integralności jest wyrażana dosłownie, część zaś jest zintegrowana z opisem innych składników schematu konceptualnego.

Różnice pomiędzy poszczególnymi podejściami nie zawsze są znaczące — niektóre modele używają różnej terminologii dla opisu tych samych pojęć; w innych modelach ta sama terminologia jest używana dla różnych pojęć.

Opisana sytuacja skłoniła organizację ISO [1]) do podjęcia prac o charakterze normalizacyjnym. Międzynarodowa grupa badawcza ISO TC97/SC5/WG3 wyodrębniła i poddała szczegółowej analizie [9] następujące trzy rodzaje podejść do konstruowania schematu konceptualnego:

- Obiekt—Atrybut—Relacja (OAR)[2])
- Binarne i N-arne Relacje Elementarne (B)
- Rachunek Predykatów (PR).

Dla każdego z nich zaproponowano projekt standardu języka opisu schematu konceptualnego. Ostatnio coraz bardziej wyodrębnia się kolejny rodzaj podejścia do modelowania konceptualnego, tzw.

- Abstrakcje Danych (AD).

Podejście to wywodzi się z podejść poprzednich, zwłaszcza z OAR i podejścia binarnego. Jednakże o ile trzy pierwsze podejścia umożliwiają w zasadzie modelowanie tylko statyki, Abstrakcje Danych uwzględniają możliwości specyfikowania funkcji i zdarzeń.

Podejście Obiekt—Atrybut—Relacja

Podstawowymi konstrukcjami modelowania, wyprowadzanymi z opracowań Chena [6] i Bachmana [2] są: obiekt, atrybut i relacja. Ich sens zawiera się w definicjach przytoczonych uprzednio. Istnieją różne warianty tego podejścia zaproponowane przez innych autorów. W podejściu OAR zawsze używa się określonej notacji graficznej, która służy jako narzędzie do tworzenia diagramów modeli konceptualnych oraz jako środek interakcji między użytkownikiem a administratorem dziedziny przedmiotowej lub administratorem bazy danych. W podejściu OAR dozwolone są relacje n-członowe (n-arne), jak również dopuszcza się przyporządkowanie relacjom atrybutów.

W tym sposobie modelowania istotne jest rozróżnienie pomiędzy obiektem, wartością atrybutu i relacją a typem obiektu, atrybutem i typem relacji. W języku opisu schematu konceptualnego OAR, zaproponowanym przez grupę ISO TC97/SC5/WG3 można wyróżnić trzy główne części (por. wydruk 1): identyfikację schematu konceptualnego (nazwa) oraz opisy typów obiektów i typów relacji. Opis typów obiektów obejmuje określenie nazw dla poszczególnych typów obiektów, ich identyfikatorów oraz nazw ich atrybutów. Podobnie, w części opisującej typy relacji specyfikuje się nazwy typów relacji, liczbę związanych typów obiektów, ich nazwy, funkcjonalną zależność między nimi, maksymalną i minimalną liczbę typów obiektów w typie relacji, identyfikator oraz atrybuty typu relacji. Przykładowy diagram OAR przedstawiono na rys. 1, a jego opis w adekwatnym języku opisu schematu konceptualnego — na wydruku 2.

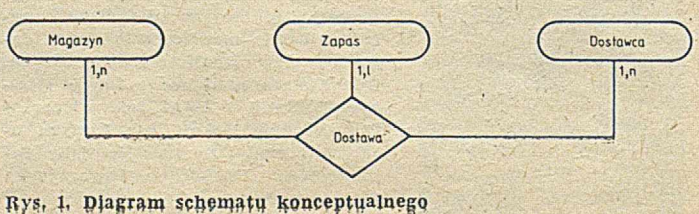
Istniejące metody modelowania bazy danych przy wykorzystaniu podejścia OAR można podzielić na dwie grupy. Pierwsza z nich — to modelowanie bezpośrednie (ad hoc), w którym twierdzenia o dziedzinie przedmiotowej pozwalają na identyfikację typów obiektów i typów relacji. Druga metoda — to modelowanie strukturalne polegające na specyfikowaniu atrybutów dziedziny przedmiotowej, a następnie konstruowaniu poszczególnych typów obiektów i typów relacji na podstawie analizy atrybutów. W tym wypadku istnieje możliwość automatycznego wykonywania niektórych czynności. Decyzja o wyodrębnieniu atrybutu czy typu obiektu jest podejmowana w trakcie iteracyjnego procesu modelowania. W odniesieniu do relacji n-częściowych jest stosowana dekompozycja umożliwiająca tworzenie relacji niższego stopnia.

```
conceptual-schema           "CONCEPTUAL SCHEMA"
                              conceptual-schema-name
                            entity-type-description
                              {entity-type-description}
                            relationship-type-description
                              {relationship-type-description}

entity-type-description  = "ENTITY-TYPE" entity-type-name
                           "IDENTIFIER" identifier
                           "DESCRIPTION"
                              attribute-type-description
                              {attribute-type-description}

relationship-type-description = "RELATIONSHIP-TYPE"
                                  relationship-type-name
                                "DIMENSION" unsigned-integer
                                "COLLECTION" entity-type-name
                                  {entity-type-name}
                                "CARDINALITY"
                                  entity-type-name minc","maxc
                                  {entity-type-name minc","maxc}
                                ["FUNCTIONAL DEPENDENCY"
                                  entity-type-name
                                  {entity-type-name}
                                  "ON" entity-type-name}
                                  [IDENTIFIERS identifier]
                                  ["DESCRIPTION
                                   attribute-type-description
                                   {attribute-type-description}]]

attribute-type-description = attribute-type-name.
minc                       = unsigned-integer|letter.
maxc                       = unsigned-integer|letter.
conceptual-schema-name     = identifying-name.
entity-type-name           = identifying-name.
identifier                 = attribute-type-name.
                               {"," attribute-type-name}.
attribute-type-name        = identifying-name.
relationship-type-name     = identifying-name.
identifying-name           = letter {letter|digit|hyphen}
```

Wydruk 1. Składnia języka opisu schematu konceptualnego dla podejścia OAR (propozycja normalizacji [9])

```
CONCEPTUAL SCHEMA DOSTAWA-TOWARU
ENTITY-TYPE ZAPAS
IDENTIFIER SYMBOL-TOWARU
DESCRIPTION SYMBOL-TOWARU
            ZAPIS-MAGAZYNOWY
            ZLOZONE-ZAMOWIENIA
            ZAPAS-POTENCJALNY
            BIEZACA-SPRZEDAZ
            BIEZACE-DOSTAWY
            BIEZACE-ZAMOWIENIA
            KOSZTY-ZAPASOW
            NORMA-ZAPASU
ENTITY-TYPE DOSTAWCA
IDENTIFIER SYMBOL-DOSTAWCY
DESCRIPTION SYMBOL-DOSTAWCY
            NAZWA-DOSTAWCY
            ADRES-DOSTAWCY
            ILOSC-DOSTARCZONYCH-ARTYKULOW
            WARTOSC-ZAMOWIENIA
ENTITY-TYPE MAGAZYN
IDENTIFIER NR-MAGAZYNU
DESCRIPTION NR-MAGAZYNU
            NAZWA-MAGAZYNU
            POWIERZCHNIA-SKLADOWA
            ZATRUDNIENIE
RELATIONSHIP-TYPE DOSTAWCA
DIMENSION     3
COLLECTION ZAPAS
           DOSTAWCA
           MAGAZYN
CARDINALITY ZAPAS      1,1
            DOSTAWCA   1,n
            MAGAZYN    1,n
FUNCTIONAL DEPENDENCY
            MAGAZYN ON ZAPAS
IDENTIFIER NR-DOSTAWY
DESCRIPTION NR-DOSTAWY
            SYMBOL-TOWARU
            ILOSC-TOWARU
            DATA-DOSTAWY
```

Wydruk 2. Opis diagramu z rys. 1 w języku OAR (por. wydruk 1)



Rys. 1. Diagram schematu konceptualnego

[1]) ISO — Międzynarodowa Organizacja Normalizacyjna (International Organization for Standardization), (przyp. red.).

[2]) Podejście to jest znane na świecie pod angielską nazwą Entity-Relationship Approach (przyp. red.).

Podejście binarne

Początki modelowania binarnego wiążą się przede wszystkim z pracami Bracchiego [4], Nijsena [13] i Falkenberga [8]. Cechy tego podejścia czynią go właściwym narzędziem użytkownika SZBD na poziomie schematu konceptualnego. Wspólną cechą różnych sposobów binarnego modelowania konceptualnego jest odzwierciedlenie dziedziny przedmiotowej przez ogół występujących w niej binarnych relacji tj. zdań, w których tylko dwa składniki odgrywają rolę. Z punktu widzenia analityka dziedzina przedmiotowa jest w każdym momencie opisywana przez zbiór obiektów i binarnych relacji między nimi. Podstawowymi konstrukcjami modelowania binarnego są zatem obiekty, nazwy obiektów i binarne relacje. Istotne znaczenie ma odróżnianie obiektów (ich wystąpień) od nazw obiektów (ich typu). W związku z tym w opisie konceptualnym wprowadza się dwa typy obiektów: obiekty niesłownikowe (ang. NOLOTS, tj. non-lexical objects) i obiekty słownikowe (ang. LOTS, tj. lexical objects). Obiekty niesłownikowe odpowiadają wystąpieniom obiektów, natomiast obiekty słownikowe nazwom obiektów. Identyfikacja obiektów (NOLOTS) następuje przez nazwy obiektów (LOTS). Zarówno słownikowe jak i niesłownikowe typy obiektów mogą być dzielone na podtypy.

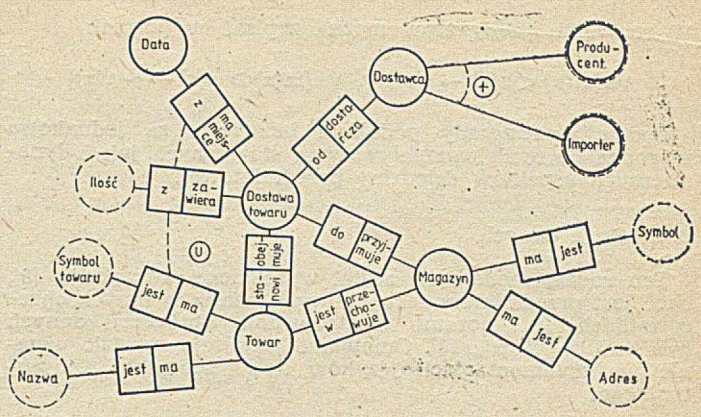
Odpowiednio do wyróżnionych dwóch typów obiektów istnieją trzy typy relacji binarnych:

- typy pojęć (ang. idea types), stanowiące powiązania między obiektami niesłownikowymi
- typy powiązań (ang. bridge types) wiążące obiekty słownikowe z określonymi przez nie obiektami niesłownikowymi
- typy wyrażeń (ang. phrase types) opisujące relacje tylko między obiektami słownikowymi.

Schemat konceptualny opisywany binarnie ma również statyczne i dynamiczne ograniczenia, będące zestawem niezbędnych twierdzeń o obiektach i relacjach. Ograniczenia, jak również relacje i obiekty są rejestrowane w schemacie konceptualnym za pomocą odpowiednich operacji wprowadzania, skreślania, przyłączania obiektów do podtypów i odłączania obiektów od podtypów, wiązania obiektów ze sobą i in. Składnia binarnego języka schematu konceptualnego (por. wydruk 3) zawiera wszystkie omówione konstrukcje, a więc obiektów słownikowych i niesłownikowych oraz relacji pojęć, powiązań i wyrażeń. Na rys. 2 przedstawiono przykładowy diagram binarnego modelu konceptualnego, a na wydruku 4 jego opis dokonany przy użyciu języka opisu binarnego schematu konceptualnego.

```
conceptual-schema = begin"
                    "CONCEPTUAL-SCHEMA called" schema-name";"
                    nolot-declaration {nolot-declaration}
                    lot-declaration {lot-declaration}
                    {subtype-declaration}
                    {idea-declaration}
                    bridge-declaration {bridge-declaration}
                    {phrase-declaration}
                    {constraint-declaration}
                    "end".
nolot-declaration = "NOLOT called" (nolot-name{;nolot-name-list}).
lot-declaration = "LOT called" (lot-name{;lot-name-list}).
subtype declaration = nolot-subtype;lot-subtype
nolot-subtype = "NOLOT called" (nolot-name-1{;nolot-name-1-list}
                "is subtype-of NOLOT called" nolot-name-2";".
lot-subtype = "LOT called" (lot-name-1{;lot-name-1-list}
              "is subtype-of LOT called" lot-name-2";".
idea-declaration = "IDEA (with-first ROLE (called" role-name-1
                   "and on NOLOT called "nolot-name-1")"
                 "and with-second ROLE (called" role-name-2
                   "and on NOLOT called "nolot-name-2"))"
                 is called "idea-name";"
bridge-declaration = "BRIDGE (with-first ROLE (called"role-name-1
                     "and on NOLOT called" nolot-name")"
                   "and with-second ROLE (called"role-name-2
                     "and on LOT called"lot-name"))"
                   "is called" bridge-name";".
phrase-declaration = "PHRASE(with-first ROLE (called" role-name-1
                     "and on LOT called" lot-name-1")"
                   "and with-second ROLE (called" role-name-2
                     "and on LOT called" lot-name-2"))"
                   "is called"phrase-name";".
constraint-declaration = "CONSTRAINT called" constraint-name
                     "is declared as "sentence";".
nolot-name-list = "{"nolot-name {nolot-name}"}"";"
lot-name-list = "{"lot-name {lot-name}"}"";"
schema-name = identifying-name.
nolot-name  = identifying-name.
lot-name    = identifying-name.
idea-name   = identifying-name.
role-name   = identifying-name.
bridge-name = identifying-name.
phrase-name = identifying-name.
identifying-name = "'" letter {letter|digit|hyphen} "'".
```

Wydruk 3. Składnia języka opisu schematu konceptualnego dla podejścia binarnego (propozycja normalizacji [9])



Rys. 2. Diagram binarnego schematu konceptualnego

```
CONCEPTUAL SCHEMA called GODPODARKA TOWAROWA
NOLOT called ('DOSTAWCA' 'DOSTAWA-TOWARU' 'TOWAR' 'MAGAZYN'
              'IMPORTER' 'PRODUCENT')
LOT called ('SYMBOL' 'ADRES' 'NAZWA' 'SYMBOL-TOWARU' 'ILOSC'
            'DATA')
NOLOT called ('PRODUCENT' 'IMPORTER')
             is subtype-of NOLOT called 'DOSTAWCA'
IDEA with-first ROLE (called 'stanowi'
                      and on NOLOT called 'TOWAR')
                and with-second ROLE (called 'obejmuje'
                      and on NOLOT called 'DOSTAWA-TOWARU')
                is called 'przesylka'
-- Pomija sie dalsze cztery deklaracje type IDEA
BRIDGE with-first ROLE (called 'ma'
                        and on NOLOT called 'MAGAZYN')
                  and with-second ROLE (called 'jest'
                        and on LOT called 'ADRES')
                  is called 'polozenie'
-- Pomija sie dalsze cztery deklaracje type BRIDGE
CONSTRAINT identyfikacja-dostawy
is declared as
   (DATA of DOSTAWA-TOWARU and ILOSC of DOSTAWA-TOWARU
   and SYMBOL-TOWARU of TOWAR is unique);
CONSTRAINT podtypy-dostawcy
is declared as
   DOSTAWCA is equal to (PRODUCENT union IMPORTER);
CONSTRAINT rozlacznosc-podtypow
is declared as
   (PRODUCENT intersection IMPORTER is empty).
```

Wydruk 4. Opis diagramu z rys. 2 w języku opisu schematu konceptualnego dla podejścia binarnego (por. wydruk 3)

W procesie modelowania binarnego schematu konceptualnego projektant wykonuje trzy zadania: ustala konwencje nazw dla obserwowanych w dziedzinie przedmiotowej obiektów, klasyfikuje je oraz przygotowuje schemat konceptualny. W tym podejściu schemat konceptualny wprowadza głównie klasyfikację i uporządkowanie w opisie dziedziny przedmiotowej, natomiast nie jest wyczerpującym opisem na poziomie wystąpień obiektów. Proces modelowania konceptualnego przy użyciu relacji binarnych, mimo prób „algorytmizacji" jest heurystyczny. Jego punktem wyjścia jest nieformalna percepcja dziedziny przedmiotowej. Po wyborze niezbędnych twierdzeń opisu dziedziny następuje wyodrębnienie obiektów słownikowych i niesłownikowych, binarnych relacji między nimi, ograniczeń, a następnie jest tworzony graficzny opis modelu konceptualnego oraz rejestracja w języku opisu schematu. W wyniku usuwania wewnętrznych niezgodności i braków w opisie, w kolejnych iteracjach dochodzi się do pożądanego poziomu jakości schematu.

Rachunek predykatów

W wypadku logiki rachunku predykatów postrzegana dziedzina przedmiotowa składa się wyłącznie z obiektów, których dotyczą twierdzenia. Schemat konceptualny stanowi opis składający się ze zbioru zdań zakodowanych w języku formalnym opartym na regułach logiki formalnej [20]. Zdania te są zbudowane z:

— nazw i zmiennych
— predykatów
— łączników logicznych
— kwantyfikatorów.

Nazwy i zmienne dotyczą obiektów w dziedzinie przedmiotowej, a zdania wyrażają twierdzenia o tych obiektach. Na przykład, zdanie:

Jeżeli wartość zapisu magazynowego m(x) obniży się poniżej zapasu zamówionego z(x), należy złożyć zamówienie t(x), w przeciwnym wypadku zamówienia nie składać.

W rachunku predykatów można zapisać w postaci:

$$\forall x[m(x) < z(x) \rightarrow t(x)] \wedge \forall x[m(x) \geq z(x) \rightarrow \sim t(x)]$$

Podstawowe zasady podejścia binarnego są jednakowo wygodne do modelowania zarówno statycznych jak i dynamicznych aspektów dziedziny przedmiotowej.

Abstrakcje danych

Ten rodzaj modeli nie został wyróżniony w raporcie [9], choć jest przedmiotem szczegółowych analiz w innych opracowaniach. Inicjatorami konstruowania abstrakcji danych byli Smithowie [17, 18]. Omawiane podejście znalazło uznanie w implementacjach języków opisu schematu konceptualnego.

Zasada tworzenia abstrakcji danych polega na usuwaniu nieistotnych cech (atrybutów) analizowanego obiektu i akceptowaniu cech adekwatnych do aktualnego kontekstu, wynikającego z analizy dziedziny przedmiotowej bądź analizy potrzeb. Cytowani autorzy wyróżnili trzy rodzaje abstrakcji danych: **klasyfikację** [19] **agregację** [17] i **uogólnienie** [18]. Wskutek stosowania tych metod abstrakcji każdy wyodrębniony w dziedzinie przedmiotowej obiekt staje się albo agregatem, albo typem obiektu rodzajowego. Skrótowo, istotę poszczególnych metod w typowaniu obiektów można przedstawić następująco:

- **klasyfikacja:** zbiór wystąpień → nowy typ
- **agregacja:** związek między typami → nowy typ (agregat)
- **uogólnienie:** zbiór typów (obiektów składnikowych) → nowy typ (obiekt rodzajowy).

W systemie ACM/PCM wprowadzono [5] podobny podział określając klasyfikację mianem asocjacji. Inni autorzy wyróżniają związki uogólnienia i podzbioru typu obiektów w ramach jednego typu obiektów [1], a jeszcze inni [16] określają trzy typy związków między typami obiektów: sumę (odpowiednik obiektu rodzajowego), produkt (agregat) i zależność (podzbiór).

Na ogół przyjmuje się, że agregacja i uogólnienie to podstawowe rodzaje abstrakcji. Odpowiadają im abstrakcje danych: agregaty i obiekty rodzajowe.

Agregacja polega na tworzeniu agregatu, tj. związku między różnymi typami obiektów. Agregat jest więc rozumiany jako zbiór typów traktowanych jak pojedynczy typ obiektu. W agregacie są pomijane niektóre cechy obiektów składnikowych, a podkreślane cechy agregatu jako całości. Na przykład, wskutek zastosowania agregacji związek:

ODBIORCA zamawia TOWAR u DOSTAWCY

między trzema typami obiektów (ODBIORCA, TOWAR i DOSTAWCA) przechodzi w nowy typ obiektu (agregat):

ZAMÓWIENIE

W procesie uogólniania następuje grupowanie typów obiektów w hierarchiczne zależności, tj. konstruowanie nowych typów obiektów rodzajowych. Na przykład, uogólnienie zastosowane do zbioru typów obiektów {**SPRZEDAWCA, MAGAZYNIER, REFERENT**} pozwala na utworzenie nowego typu obiektu rodzajowego **PRACOWNIK**. Typ obiektu rodzajowego **O** jest uogólnieniem typów obiektów składowych O_1, O_2... O_n, jeżeli każde wystąpienie typu obiektu **O** jest jednocześnie wystąpieniem przynajmniej jednego z typów obiektów O_1, O_2... O_n.

Do opisu agregatów i obiektów rodzajowych można zastosować prosty język deklaracji obiektów, zaproponowany w [17, 19]. Utworzony wyżej agregat **ZAMÓWIENIE** można w tym języku zadeklarować jako:

```
type zamówienie — agregate O, T, D
                  O: odbiorca
                  T: towar
                  D: dostawca
                  end
```

Obiekty rodzajowe opisuje się w sposób nieco bardziej złożony. W ich definicji **(def)** specyfikuje się obiekty składowe **(subs)** i atrybuty **(coms)**, np.:

```
def pracownik: sub
                  sprzedawca, magazynier
               com
                  nazwisko, imię, płaca, staż
               end
```

Obiekty składowe obiektu rodzajowego, w tym wypadku sprzedawca i magazynier, muszą być opisane w oddzielnym zdaniu **def**. Jeśli są to typy proste, to odpowiadające im zdania **def** nie zawierają części **sub**, lecz jedynie specyfikację atrybutów **com**.

LITERATURA

[1] Atzeni P. et al.: INCOD-DTE — A System for Interactive Conceptual Design of Data, Transactions and Events. Pp. 205-228, Ceri S. (ed.): Methodology and Tools for Data Base Design, North Holland, Amsterdam 1983

[2] Bachman C. W.: Data Structure Diagrams. Pp. 108—116, File Organization, Selected Papers from File 68 IAG Conf., Swetz and Zeitlinger (eds.), Amsterdam 1969

[3] Biller H., Neuhold E. J.: Concepts for Conceptual Schema. Nijssen G. M. (ed.): Architecture and Models in Data Base Management Systems, North Holland, Amsterdam 1977

[4] Bracchi G., Paolini P., Pelagatti G.: Binary Logical Associations in Data Modelling. Pp. 125—148, Nijssen G. M. (ed.): Architecture and Models in Data Base Managament Systems, North Holland, Amsterdam 1977

[5] Brodie M. L., Silva E.: Active and Passive Component Modelling ACM/PCM. Pp. 41—91, Olle T. W., Sol G. H., Verrijn-Stuart A. A.: Information Systems Design Methodologies: A Comparative Review, North Holland, Amsterdam 1982

[6] Chen P. P. S.: The Entity-Relationship Model — Toward a Unified View of Data. ACM Trans. on Database Systems, Vol. 1, No. 1, March 1976

[7] Davis C. G. et al. (eds.): Entity-Relationship Approach to Software Engineering. 3rd Intern. Conf. on Entity-Relationship Approach, Anheim (CA) USA, 5—7 october 1983, North Holland, Amsterdam 1983

[8] Falkenberg E. D.: Concepts for Modelling Information. Pp. 95—109, Nijssen G. M. (ed.): Modelling in Database Management Systems, North Holland, Amsterdam 1976

[9] Griethuysen J. J. et al. (eds.): Concepts and Terminology for the Conceptual Schema and Information Base. Report ISO TC97/ /SC5/WG3, February 1982

[10] Kerschberg L., Klug A., Tsichritzis D.: A Taxonomy of Data Models. Pp. 43—64, Lockeman P. C., Neuhold E. J. (eds.): Systems for Large Data Bases, North Holland, Amsterdam 1977

[11] Kung C. H.: An Analysis of Three Conceptual Models with Time Perspective. Pp. 141—167, Olle T. W., Sol H. G., Tully C. J.: Information Systems Design Methodologies — A Feature Analysis, North Holland, Amsterdam 1983

[12] Lenzerini M., Santucci G.: Cardinality Constraints in the Entity-Relationship Model. Pp. 529—550, Davis C. G. et al. (eds.): Entity-Relationship Approach to Software Engineering, North Holland, Amsterdam 1983

[13] Nijssen G. M.: A Gross Architecture for the Next Generation Database Management Systems. Pp. 1—24, Nijssen G. M. (ed).: Modelling in Data Base Management Systems, North Holland, Amsterdam 1976

[14] Olive A.: Analysis of Conceptual and Logical Models in Information Systems Design Methodologies. Pp. 63—85, Olle T. W., Sol H. G., Tully C. J.: Information Systems Design Methodologies — A Feature Analysis, North Holland, Amsterdam 1983

[15] Pszczołowski T.: Mała encyklopedia prakseologii i teorii organizacji, Ossolineum, Wrocław 1978

[16] dos Santos C. S., Neuhold E. J., Furtado A. L.: A Data Type Approach to the Entity-Relationship Model. Pp. 105—119, Chen P. P. S. (ed.): Entity-Relationship Approach to Systems Analysis and Design, North Holland, Amsterdam 1980

[17] Smith J. M., Smith D. C. P.: Database Abstractions — Aggregation. Comm. of the ACM, Vol. 20, No. 6, pp. 405—412, June 1977

[18] Smith J. M., Smith D. C. P.: Database Abstractions — Aggregation and Generalization. ACM Trans. on Database Systems, Vol. 2, No. 2, pp. 105—133, June 1977

[19] Smith D. C. P., Smith J. M.: Conceptual Database Design. Pp. 137—162, Data Design-Infotech State of the Art Report, Vol. 1—2, 1980

[20] Steel T. B. Jr: Report on ISO Activity in Data Base Standardization. Pp. 84—91, Dean S. M., Hammersley P. (eds.): Proc. Intern. Conf. on Data Bases, University of Aberdeen, July 1980.

JAN BIELECKI
Instytut Informatyki
Politechnika Warszawska

Język C implementacja Lattice/Microsoft

Kompilator języka C opracowany przez firmę Lattice, a rozpowszechniony zarówno przez Lattice jak i Microsoft, należy do najczęściej używanych kompilatorów języka C dla mikrokomputera IBM PC. Swoją pozycję zawdzięcza on wysokiej jakości generowanego kodu oraz niemal wiernemu dostosowaniu się do wymogów języka wzorcowego.

Tym niemniej — przy bliższej analizie — okazuje się, że omawiany kompilator wykazuje pewne odstępstwa od standardu [1]. Odstępstwa te są nieznaczne, mają w wielu wypadkach charakter rozszerzeń i ulepszeń, jednak ich znajomość jest niezbędna z punktu widzenia diagnostyki kompilacyjnej i przenośności oprogramowania. Cyfry w nawiasach przy poszczególnych tytułach, poniżej, oznaczają odpowiednie rozdziały podręcznika [1].

KOMENTARZE (2.1)

Komentarze są traktowane jako zagnieżdżone. Oznacza to, że w tej implementacji poprawnym komentarzem jest napis:

```
/* pierwszy-poziom/* drugi poziom */ pierwszy-poziom */
```

mimo iż w języku wzorcowym składa się on z komentarza:

```
/* pierwszy-poziom /* drugi-poziom */
```

oraz napisu:

```
pierwszy-poziom */
```

Jeśli kompilacja zostanie wykonana z opcją "—c", to nie będzie brane pod uwagę zagnieżdżanie komentarzy.

Przykład. Program:

```
main () {
  /*/**/ print("jb"); /*/
  }
```

zostanie uznany za poprawny jedynie wtedy, gdy będzie skompilowany z opcją "—c".

IDENTYFIKATORY (2.2)

Poprawny identyfikator może zawierać, oprócz liter, cyfr i symboli podkreślenia, także znaki "$". Znak ten nie może być pierwszym znakiem identyfikatora.
Przykład. Deklaracja:

```
char ascii$;
```

wprowadza identyfikator **ascii$** różny od identyfikatora **ascii.**

LITERAŁY ZNAKOWE (2.4.3)

Literały znakowe mogą liczyć więcej niż jeden znak. W zależności od liczby użytych znaków literały takie są typu **(int)** albo **(long int).** Jeśli literał zawiera kreskę ukośną pochyloną w lewo (ukośnik), po której bezpośrednio występuje mała litera x, to następne znaki literału są traktowane jako cyfry szesnastkowe.

Przykład. Opracowanie deklaracji:

```
int vara = '56';
    varb = '|x56';
```

ma taki skutek jak opracowanie deklaracji:

```
int vara '5'<<8+'6'
    varb=5<<4+6;
```

LITERAŁY TEKSTOWE (2.5)

Literały tekstowe nie mogą liczyć więcej niż 256 bajtów, a wszystkie literały tekstowe o tej samej wartości są reprezentowane za pomocą jednej stałej tekstowej.

Przykład. Wykonanie programu:

```
main () {
  printf(*"a"++);
  printf("a");
}
```

powoduje wprowadzenie napisu **ab.** W języku wzorcowym byłby to napis **aa.**

UWAGI IMPLEMENTACYJNE (2.6)

W tabeli 1 zestawiono właściwości danych różnych typów.

Tabela 1. Właściwości danych różnych typów

Typ danej	Rozmiar w bajtach	Zakres wartości
char	1	Od 0 do 255
int	2	Od —32 768 do 32 767
short int	2	Od —32 768 do 32 767
unsigned	2	Od 0 do 65 535
long int	4	Od $-2*10^9$ do $2*10^9$
float	4	0 oraz od -10^{38} do -10^{-37} i od 10^{-37} do 10^{38}
double	8	0 oraz od -10^{308} do -10^{-307} i od 10^{-307} do 10^{308}
Wskazujący	2 albo 4	0 oraz wartości wskazujące

W modelach pamięci S i P dane wskazujące mają rozmiar 2 bajty, a w modelach D i L 4 bajty. Wybór modelu jest podyktowany przewidywanym rozmiarem przestrzeni adresowej dla programu i danych, zgodnie z zestawieniem podanym w tabeli 2.

Tabela 2

Model	Przestrzeń programu	Przestrzeń danych
S	Do 64 KB	Do 64 KB
P	Do 1 MB	Do 64 KB
D	Do 64 KB	Do 1 MB
L	Do 1 MB	Do 1 MB

WYRAŻENIA PIERWOTNE (7.1)

Dla każdego typu struktury oraz unii utrzymywana jest przez kompilator odrębna lista nazw komponentów, a w odwołaniach do nich konieczne jest z tego względu jednoznaczne zidentyfikowanie agregatu. Dlatego wyrażenie pierwotne występujące przed operatorem "." i "—>" musi identyfikować określony agregat albo wskazanie na określony agregat.

Przykład. Program w języku wzorcowym:

```
main () {
  struct tag {
    int var;
  } str;
  int *ptr;
  ptr = (int *) & str;
  ptr—>var = 5;
  printf("%d", str.var);
}
```

jest w tej implementacji uznawany za niepoprawny, gdyż w drugiej instrukcji przypisania nie można jednoznacznie zidentyfikować agregatu. Przystosowanie tej instrukcji do wymagań kompilatora można uzyskać nadając jej postać:

```
( (struct tag *)ptr)—>var = 5;
```

OPERATORY JEDNOARGUMENTOWE (7.2)

W omawianej implementacji osłabiono wymaganie, aby operator referencji dotyczył 1-wyrażenia. Przyjęto, że może on dotyczyć także nazwy tablicy, która jak wiadomo nie jest 1-wyrażeniem. Rezultatem referencji jest w takim wypadku wskazanie na tablicę, istotnie różne od wskazania jej pierwszego elementu.

Przykład. Program w języku wzorcowym, o postaci:

```
char arr [2] [3],
     (*ptr) [2] [3];
main () {
  ptr = (char (*)[2][3])arr;
  ...
}
```

może być w implementacji uproszczony do postaci:

```
char arr [2] [3],
     (*ptr) [2] [3] =&arr;
main () {
  ...
}
```

OPERATORY RELACJI (7.6)

Jeśli w implementacji następuje wyznaczenie relacji danych wskazujących różnych typów, to prawy argument zostaje poddany konwersji na typ lewego argumentu. Jeśli następuje wyznaczenie relacji danej całkowitej i danej wskazującej, to dana całkowita zostaje poddana konwersji na typ danej wskazującej. W większości wypadków operacje takie nie mają sensu i z tego względu należy ich unikać.

OPERATORY PORÓWNAŃ (7.7)

Jeśli w implementacji następuje porównanie danych wskazujących różnych typów, to prawy argument zostaje poddany konwersji na typ lewego argumentu. Jeśli następuje porównanie danej całkowitej i danej wskazującej, to dana całkowita zostaje poddana konwersji na typ danej wskazującej. Podobnie jak w wypadku relacji sens takich operacji jest wątpliwy.

OZNACZENIA KLASY PAMIĘCI (8.1)

W implementacji nie wykorzystano możliwości optymalizacyjnych, jakich dostarcza klasa **register**. Mimo to zachowano ograniczenie wynikające z użycia zmiennych tej klasy.

Przykład. Błędny program w języku wzorcowym i w implementacji

```
main () {
  register var = 5;
  int *ptr =&var;
  printf("%d", *ptr);
};
```

DEKLARACJE STRUKTUR I UNII (8.5)

Dla każdego typu struktury oraz unii utrzymywana jest w tej implementacji odrębna lista nazw składowych komponentów, a więc nie zachodzą kolizje nazw składowych komponentów oraz nazewników różnych struktur i unii.

Przykład. Błędny program w języku wzorcowym:

```
main () {
  struct {
    int var;
    char chr;
  } str1;
  struct {
    char chr;
    int var;
  } str2;
  ...
}
```

jest w implementacji programem poprawnym.

NAZWY TYPÓW (8.7)

W implementacji zabroniono, aby nazwy typów dotyczyły nie zadeklarowanych jeszcze struktur albo unii, tj. postawiono wymaganie, aby w takim wypadku powołać się na uprzednio zadeklarowany nazewnik.

Przykład. Operator konwersji o postaci:

```
(struct{float re,im;} *)
```

powinien być zastąpiony, na przykład, operatorem

```
(struct tag *)
```

gdzie tag jest nazewnikiem uprzednio zadeklarowanym jako

```
struct tag {
  float re,im;
};
```

DEFINICJE FUNKCJI (10.1)

W implementacji przyjęto, że jeśli wynik funkcji ma być typu **(char)** albo **(short int)**, to jest typu **(int)**. Podobnie przyjęto, że jeśli wynik funkcji ma być typu **(float)**, to jest typu **(double)**.

Przykład. W następstwie wykonania programu:

```
char fun ();
main () {
  printf("%d",fu() );
}
char fun (){
  return 259;
}
```

w omawianej implementacji jest wyprowadzona liczba 259 natomiast zgodnie z językiem wzorcowym powinna być wyprowadzona liczba 3.

KOMPILACJA WARUNKOWA (12.3)

W omawianej implementacji egzekwowane jest wymaganie, aby wyrażenie stałe występujące w dyrektywie kompilacji warunkowej **#if** nie zawierało operatora **sizeof**.

OGRANICZENIA I WŁAŚCIWOŚCI IMPLEMENTACJI

1. Rozmiar zadeklarowanego obiektu nie może przekraczać 32 767 bajtów. Sumaryczny rozmiar zadeklarowanych obiektów tej samej klasy nie może przekraczać także 32 767 bajtów.
2. Maksymalna wartość wyrażenia indeksującego tablicę nie może przekraczać 32 766.
3. Rozmiar literału tekstowego nie może przekraczać 256 bajtów.
4. Liczba parametrów makrodefinicji nie może przekraczać 8. Podczas wywoływania takiej makrodefinicji cały napis wywołujący musi mieścić się w jednym wierszu.
5. Liczba zagnieżdżeń podczas wykonywania dyrektywy **#include** nie może przekraczać 8.
6. Pamięć dla zmiennych lokalnych bloku jest przydzielona w prologu funkcji obejmującej deklarację tych zmien-

nych. Powoduje to przydzielenie pamięci z pewnym nadmiarem, jednak w zgodzie z [1].

7. Wyznaczanie wartości wyrażeń przebiega ściśle w kolejności od lewej do prawej strony (oczywiście z uwzględnieniem priorytetów). Wyjątek od tej reguły dotyczy jedynie wspólnych podwyrażeń, na przykład takich jak w instrukcji

a [i+2] = b[i+2];

Podwyrażenia takie są rozpoznawane, a ich wartości wyznaczane jednokrotnie.

LITERATURA

[1] Kerninghan B. W., Ritchie D. M.: The C programming language. Prentice Hall, Englewood Cliffs (NJ), 1978

[2] Lattice Inc.: Lattice 8086 C Compiler — Functional Description Manual, 1983

[3] Microsoft Corp.: Microsoft C Compiler for 8086 and 8088 microprocessors. Bellévue (WA), 1984.

JACEK AMBROZIAK
Środowiskowe Centrum Obliczeniowe
CYFRONET
Instytut Energii Atomowej
Otwock-Świerk

Systemy ekspertowe (2)

W drugiej części artykułu skoncentrowano się na nietechnicznych aspektach rozwoju systemów ekspertowych, przedstawiając ich stosunek do metod klasycznych oraz możliwości rozwoju tych systemów.

PORÓWNANIE Z KLASYCZNYMI METODAMI PRZETWARZANIA INFORMACJI

Technologia systemów ekspertowych wnosi coś istotnie nowego do metod stosowania środków informatycznych. Systemy te w wielu aspektach różnią się od programów tworzonych klasycznymi metodami algorytmicznego przetwarzania informacji. Te właśnie różnice stanowią o możliwościach stosowania systemów ekspertowych tam, gdzie podejście algorytmiczne jest nie do pomyślenia z uwagi na pojęciową złożoność zadania.

Charakterystyczny jest zwłaszcza sposób wykorzystania wiedzy, mianowicie operowanie jej jawną reprezentacją. Klasyczne metody programowania wykorzystują specyficzną wiedzę o problemie już w czasie projektowania algorytmu. Prowadzi to do konstrukcji programów wprawdzie efektywnych, ale mało elastycznych i jednostronnych.

W programie klasycznym użyta wiedza jest na stałe związana z algorytmem. W systemie ekspertowym mamy do czynienia z inną sytuacją. Częścią aktywną jest program operujący symboliczną reprezentacją wiedzy. Jest on w zasadzie zupełnie niezależny od merytorycznej zawartości bazy wiedzy; jest natomiast ukierunkowany na wybrany formalizm reprezentacji.

Zalety odseparowania wiedzy specyficznej dla danej klasy zadań od programu stosującego tę wiedzę są następujące:

- baza wiedzy podlega w zasadzie łatwej inspekcji — łatwo można się zorientować „o czym program wie". W programach klasycznych wiedza jest „zaszyta" w algorytmie i tylko szczegółowa jego analiza lub lektura dokumentacji pozwala się dowiedzieć, z jakich faktów i sposobów postępowania (specyficznych dla danej dziedziny zastosowania) program korzysta. W tym sensie program z bazą wiedzy zawiera ważną część swojej własnej dokumentacji;
- wiedza jest reprezentowana w systemie ekspertowym czytelnie i w formie zmodularyzowanej. Względnie łatwo można więc lokalizować określone jej fragmenty, zmieniać je, dodawać nowe — co jest szczególnie ważne podczas uruchamiania programu;
- w systemie ekspertowym wiedza może być łatwo wykorzystana na wiele sposobów. W szczególności, łatwo można zastosować ją w procesie wyjaśniania użytkownikowi akcji programu; sprowadza się to do odpowiedniego cytowania stosownych fragmentów bazy wiedzy. Dodanie wyjaśnień do programów klasycznych nie wydaje się tak proste;
- niezależność używanej wiedzy od jej reprezentacji i od mechanizmów manipulujących nią stwarza interesującą możliwość. Okazuje się, że jest możliwe „przeprogramowanie" systemu ekspertowego na inną dziedzinę zastosowań przez prostą wymianę bazy wiedzy, tzn. przy zachowanej strukturze następuje zmiana treści reguł. W ten sposób powstał z systemu MYCINA system diagnozujący choroby płuc — PUFF.

Co ciekawsze, ta sama baza może być użyta przez dwa różne systemy ekspertowe, stawiające sobie różne cele. Tak więc baza wiedzy MYCINA została wykorzystana w systemie GUIDON, przeznaczonym do nauczania diagnozowania chorób zakaźnych. Obie opisane możliwości można wykorzystać łącznie — baza wiedzy systemu PUFF po przeniesieniu do systemu GUIDON tworzy system nauczający diagnostyki pulmonologicznej.

MOTYWACJE ROZWOJU

Dziedzina systemów ekspertowych weszła w latach osiemdziesiątych w okres błyskotliwej kariery. Z jednej strony sama idea systemu ekspertowego jest bardzo pociągająca z drugiej — pojawiły się możliwości jej coraz pełniejszego urzeczywistniania. Kolosalna użyteczność systemów ekspertowych, które zgodnie ze swym głównym założeniem ideowym mają automatycznie stosować heurystyczną wiedzę specjalistyczną, wynika choćby z następujących obserwacji:

- we wszystkich dziedzinach życia odczuwa się raczej brak niż nadmiar kompetentnych specjalistów, szczególnie w krajach słabiej rozwiniętych i poza ośrodkami akademickimi. Systemy ekspertowe mogą pokryć część tego zapotrzebowania;
- wykształcenie specjalisty jest kosztowne i czasochłonne, wymagające wielu lat nauki i praktyki; kosztowne jest także korzystanie z jego usług, o ile w ogóle można je sobie zapewnić. Systemy ekspertowe są więc opłacalne ekonomicznie;
- tworzenie baz wiedzy pozwala obiektywizować i zapisywać — skądinąd prawie niedostępną — prywatną wiedzę heurystyczną specjalistów.

Wymienione problemy i proponowane rozwiązania dotyczą przede wszystkim roli systemów ekspertowych jako systemów konsultacyjnych. Technologia automatycznego przetwarzania wiedzy specjalistycznej może znaleźć także inne zastosowania. Można podać co najmniej dwa przykłady z tego zakresu.

Kompleksowa automatyzacja procesów produkcyjnych może wymagać heurystycznego podejmowania decyzji w warunkach niepełnej informacji i w czasie rzeczywistym. Dziś decyzje takie podejmują ludzie posługujący się zazwyczaj niesformalizowaną wiedzą, określaną mianem wyczucia lub intuicji. W przyszłości aparatura automatyki może być rozszerzona o systemy wykorzystujące specjali-

styczną wiedzę o procesie technologicznym oraz wiedzę zdroworozsądkową.

Skomplikowana aparatura pomiarowa dostarcza wielu strumieni informacji o różnych aspektach mierzonych zjawisk. Odpowiedni specjalista interpretuje te dane, tzn. przypisuje im określone znaczenia. Mimo iż stereotypowe, zadanie to wymaga wiedzy o tym, co i w jaki sposób jest mierzone. Technologia systemów ekspertowych może poważnie ulepszyć narzędzia pomiarowe. Po pierwsze, informację wyjściową można zorganizować w sposób odpowiedni, zrozumiały dla praktyka danej dziedziny. Po drugie, wiedza może być wykorzystana w samym procesie pomiarowym: wystąpienie nieoczekiwanych danych może spowodować automatyczne powtórzenie pomiaru, wybór innej metody pomiaru, wybór innych parametrów następnego pomiaru itd.

Systemy ekspertowe mają więc do odegrania wiele ról — są naprawdę potrzebne. Tę potrzebę wzmacniają szybkie postępy informatyki, szczególnie w sferze sprzętu. Gdy dziedzina ta powstawała, jedyny dostępny sprzęt liczący stanowiły wielkie i drogie komputery zlokalizowane w centrach obliczeniowych ośrodków akademickich. Szerokie wykorzystanie powstającej technologii nie było możliwe. Dziś, wraz z nadejściem ery mikroinformatyki sytuacja zmieniła się diametralnie — produkuje się masowo komputery osobiste przewyższające mocą obliczeniową sprzęt dostępny pionierom sztucznej inteligencji. Pojawiają się też nowe urządzenia programowe, kumulujące doświadczenia na temat reprezentacji wiedzy, struktury procesów wnioskowania symbolicznego i ogólnie — architektury systemów ekspertowych. Narzędzia te pozwalają konstruktorom systemów skupić się na zagadnieniach przetwarzania wiedzy i uwolnić się od rozważania szczegółów czysto informatycznych. Przyspiesza to wydajnie proces konstrukcji tych systemów, a także zachęca nieinformatyków do wzięcia udziału w ich tworzeniu.

TRUDNOŚCI

Możliwych, wartościowych zastosowań systemów ekspertowych jest bardzo wiele, a sama ich najogólniejsza idea jest prosta — forma przetwarzanej informacji jest czytelna nie tylko dla informatyków (inaczej niż, na przykład, w wypadku systemów operacyjnych lub kompilatorów). Te i inne zalety technologii przetwarzania wiedzy mogą łatwo doprowadzić do przekonania, że budowa praktycznego systemu nie powinna sprawiać większych trudności. Literatura popularyzująca tę tematykę przedstawia — zazwyczaj powierzchownie — systemy już zbudowane, najczęściej regułowe. Mechanizm użycia reguł jest zrozumiały i sposób reprezentacji wiedzy w przykładowych regułach wydaje się oczywisty. Jednakże mimo niezaprzeczalnych zalet systemów ekspertowych i warunków sprzyjających ich konstrukcji, praktyka tej dziedziny jest wciąż w wielkiej mierze sztuką, daleką od działalności rutynowej. Systemów sprawdzonych i powszechnie uznanych jest wciąż niewiele. W literaturze przedmiotu najczęściej cytuje się te właśnie programy (wymienione również w niniejszym opracowaniu). Jednak tylko nieliczne spośród nich są stosowane w praktyce zgodnie z przeznaczeniem.

Poniżej podjęto próbę wstępnego zanalizowania źródeł podstawowych trudności, które muszą pokonywać twórcy systemów ekspertowych.

Zdobywanie wiedzy

Najważniejszą częścią systemu ekspertowego jest baza wiedzy. Utworzenie jej jest najbardziej pracochłonnym etapem budowy systemu. Rozmaite formy automatycznego tworzenia bazy wiedzy są wciąż jeszcze słabo rozwinięte i standardem jest tu — nieodmiennie od 20 lat — postępowanie wymagające długiej i żmudnej współpracy specjalisty reprezentującego dziedzinę zastosowania systemu ekspertowego z inżynierem (ang. knowledge engineer) zajmującym się jego funkcjonowaniem z informatycznego punktu widzenia.

Na początku muszą oni wypracować wspólne rozumienie projektowanego systemu, w szczególności jak najdokładniej określić klasę zadań, które system ma rozwiązać. Jest to tym trudniejsze, że zazwyczaj współpracującym stronom brak wspólnego języka — ekspert jest laikiem w dziedzinie metod informatyki, a konstruktor systemu ekspertowego nie zna się na wybranej dziedzinie zastosowania. Ich spojrzenia na system ekspertowy będą zawsze różniły się — w praktyce jednak zazwyczaj dochodzą oni do porozumienia wystarczającego do efektywnego rozwiązania podjętego zadania.

Nawet wtedy jednak, gdy ekspert zrozumie założenia i kwestie szczegółowe stosowanej metodologii przetwarzania wiedzy, może mieć wielkie trudności z zapisaniem swych wiadomości i umiejętności w wybranym schemacie reprezentacji. To, co czyni go ekspertem, polega na zautomatyzowaniu wielu drobnych kroków wnioskowania w większe całości, stosowane szybko i bez zastanowienia. Na tej samej zasadzie można chodzić, rozmawiać itp., nie myśląc o szczegółach tych skądinąd skomplikowanych czynności. Przy konstruowaniu bazy wiedzy wszystkie istotne szczegóły procesu wnioskowania eksperta muszą być określone i odpowiednio zapisane. Zadanie to uważa się za bardzo trudne. Praca nad projektem systemu ekspertowego może być dla eksperta pierwszą i jedyną okazją uświadomienia sobie szczegółów własnego myślenia.

Mimo zalet stosowanych dziś metod reprezentacji (czytelność, modularność), tworzenie dużych baz wiedzy napotyka trudności związane z nieprzewidzianymi interakcjami częściowych segmentów wiedzy — pojawia się bariera złożoności. Aby sprostać tym zadaniom, tworzy się systemy śledzące wykorzystanie wiedzy w programie i umożliwiające ekspertowi wykonywanie niezbędnych poprawek. Programem najbardziej znanym pod tym względem jest TEIRESIAS, opracowany początkowo dla MYCINA, a przez to stosowalny do innych, podobnych systemów ekspertowych. Programy te są pewnego rodzaju specjalizowanymi edytorami baz wiedzy, wrażliwymi na składnię i w coraz większym stopniu na semantykę zapisanej formalnie wiedzy. Środki te pozwalają znacznie skrócić czas budowy bazy wiedzy, ale nie sprowadzają jeszcze tego zadania do poziomu postępowania rutynowego.

Wybór dziedziny

Technologia systemów ekspertowych stanowi narzędzie o wielkim potencjale, lecz wciąż bardzo ograniczone ze względu na możliwości zastosowania. Kluczowy dla projektu systemu ekspertowego jest wybór odpowiedniej dziedziny zastosowania, a w niej — właściwej klasy zadań. Niełatwo z góry określić, jakie zadania dadzą się rozwiązać przy użyciu dzisiejszych technik sztucznej inteligencji (powstaje tu więc naturalna sposobność doskonalenia tych technik). Z pewnością dadzą się rozwiązać zadania dostatecznie podobne do już rozwiązanych; jest tu miejsce na rozwój wszerz, tj. wykorzystanie już sprawdzonych metod. W praktyce systemów ekspertowych znaczną rolę odgrywa sztuka wyboru dziedziny zastosowania. Doświadczenie podsuwa tu konstruktorom wiele użytecznych wskazówek heurystycznych, np. rozważać zadania, które:

- ekspert może rozwiązać przez telefon, a więc nie wymagające znaczącego udziału zmysłów (czy innych dyspozycji o fizycznym charakterze), lecz informacji zwerbalizowanej,
- ekspert rozwiązuje w czasie od 3 minut do 3 godzin,
- nie prowadzą do poważnych różnic zdań przy rozwiązywaniu przez kilku ekspertów,
- wymagają przede wszystkim wnioskowania symbolicznego (nie zaś, na przykład, obliczeń numerycznych),
- nie mają rozwiązań algorytmicznych, które byłyby zazwyczaj efektywniejsze.

Reprezentacja wiedzy

Stosowane obecnie formalizmy reprezentacji wiedzy, mimo dużego wyboru, są dalekie od doskonałości. Brakuje zadowalających metod reprezentacji zjawisk zmiennych w czasie oraz złożonych stosunków przestrzennych. Niełatwo więc programować automatyczne rozwiązywanie zadań wymagających od człowieka znaczącego posługiwania się wyobraźnią przestrzenną.

Wiedza o jakiejś dziedzinie, pozwalająca skutecznie rozwiązywać praktyczne jej problemy, jest wielopostaciowa. Występują w niej elementy statyczne — o znanych faktach, o zależnościach między faktami itp., jak i dynamiczne — o strategiach rozwiązywania problemów. Formalizm reprezentacji wiedzy powinien mieć odpowiednią siłę ekspresji, wystarczającą do zapisania wszystkich potrzebnych informacji bez uciekania się do sztuczek pogarszających czytelność bazy wiedzy i utrudniających dalsze prace nad systemem. Wymaganie to wpływa na coraz większą złożoność i przez to trudność użycia najlepszych współczesnych systemów reprezentacji wiedzy.

Użyteczność

Celem technologii systemów ekspertowych jest budowa nowych, użytecznych narzędzi. Działający bez zarzutu sy-

stem, kompetentny w swojej dziedzinie, spełnia podstawowy i konieczny warunek użyteczności. Okazuje się jednak, że nie jest on wystarczający. System musi spełniać kilka dodatkowych wymagań, aby mógł być rzeczywiście wykorzystywany przez użytkowników, dla których został zaprojektowany.

Przedstawienie użytkownikowi gotowego rozwiązania postawionego zadania zazwyczaj nie wystarcza do jego zaakceptowania, skoro nie wiadomo, w jaki sposób rozwiązanie zostało znalezione. System eksportowy musi więc udzielić użytkownikowi potrzebnych wyjaśnień, dotyczących szczegółów drogi wiodącej od danych do rozwiązania. Problem wiarygodności rozwiązania jest szczególnie widoczny w dziedzinie diagnostyki medycznej. Na szczęście, jak już wspomniano, jawna reprezentacja wiedzy umożliwia względnie łatwe technicznie rozwiązanie problemu wyjaśniania. W wypadku bardziej złożonych systemów wyjaśnianie polegające na opisywaniu aktualnego kontekstu wnioskowania może okazać się niewystarczające. Rozwiązanie tej kwestii stanowi problem sam w sobie i wymaga zaprojektowania odpowiednich strategii wyjaśniania, na przykład wykorzystujących model użytkownika (jakie jest jego fachowe przygotowanie, co wie o obecnej konsultacji), poza tym różne poziomy abstrakcji stosowanej wiedzy itp.

Innym wymiarem użyteczności systemu jest właściwa organizacja dialogu system-użytkownik. Pożądane jest zastosowanie języka naturalnego (co nie musi być bardzo trudne w wąsko specjalizowanych zadaniach) oraz interakcyjnej grafiki.

Wreszcie, wielką wagę posiada właściwe określenie roli, którą ma odgrywać system tam, gdzie ma być rutynowo stosowany. Przykładu właściwych rozwiązań dostarczają systemy PUFF (diagnostyka pulmonologiczna) i ONCOCIN (chemioterapia nowotworów). Są to systemy konsultacyjne, w których forma wyprowadzania informacji została dopasowana do istniejących zwyczajów prowadzenia szpitalnej dokumentacji chorób. Lekarz nie zawsze chętnie przystanie na „wysłuchanie" komputerowych rad, zawsze jednak z radością pozbędzie się „roboty papierkowej" (która też wymaga wiedzy specjalistycznej). Raporty produkowane przez program PUFF akceptowane najczęściej bez zastrzeżeń, są co najwyżej dodatkowo komentowane i podpisywane przez lekarzy.

W drugim systemie przyjęto interesujący model konsultacji, opierający się na krytykowaniu planów terapeutycznych tworzonych przez lekarza. ONCOCIN tylko wtedy informuje o rezultatach swego wnioskowania, gdy stwierdzi, że różnią się one znacznie od propozycji lekarza. Inicjatywa należy więc stale do lekarza, co sprzyja akceptowaniu systemu. System interweniuje tylko wtedy, gdy jest to naprawdę potrzebne. Jeśli bowiem plany chemioterapii stworzone niezależnie przez człowieka i przez program różnią się w istotnym stopniu, to w grę wchodzą dwie możliwości:

- istnieje błąd w bazie wiedzy systemu i należy go usunąć,
- lekarz zapomniał o jakimś ważnym szczególe rozważanej sytuacji.

W obu wypadkach lekarz musi być poinformowany o rozbieżności.

* * *

Systemy eksportowe są niezwykle ekscytującą dziedziną współczesnej techniki — obiecują bardzo wiele, jednak za cenę rozwiązania wielu trudnych problemów teoretycznych, praktycznych i do pewnego stopnia nawet socjologicznych. Nie istnieje żadna teoria systemów ekspertowych. Istnieje natomiast ogólna ideologia tych systemów i w jej ramach kolejne, coraz doskonalsze dokonania praktyczne. W niniejszym artykule starano się przedstawić elementy tej ideologii.

Istnieje już dziś bogata literatura tego przedmiotu, w przeważającej mierze składająca się z krótkich doniesień o losach prac nad różnymi programami. Prac o ogólniejszym charakterze jest bardzo mało. Wszystkie te materiały starzeją się w błyskawicznym tempie. Czytelnikom, którzy zechcą dowiedzieć się czegoś więcej o systemach ekspertowych, warto polecić przestudiowanie literatury [1—4]. Bezsprzecznie najlepszą metodą poznania omawianej dziedziny jest jednak rzeczywiste budowanie systemów ekspertowych. Skoro pozostaje ono wciąż sztuką, trudno przecenić wartość praktycznego doświadczenia.

LITERATURA

[1] Devis R., Lenat D.: Knowledge-Based Systems in Artificial Intelligence, McGraw-Hill, New York, 1982
[2] Hayes-Roth F. et al.: Building Expert Systems. Addison-Wesley, Reading (MA), 1983
[3] Infotech Ltd.: Expert Systems — State of the Art Report, Vol. 12:7, Pergamon Press, 1984
[4] Weiss S., Kulikowski C.: A Practical Guide to Designing Expert System. Chapman and Hall, 1984.

ROMAN GRABOWICZ
JANUSZ ZALEWSKI
Warszawa

System operacyjny PC-DOS (4)

W kolejnym odcinku zewnętrznego opisu systemu operacyjnego PC-DOS przedstawiono dodatkowe polecenia wprowadzone w wersji 3.0, omówiono dostępne dla użytkownika możliwości konfigurowania systemu operacyjnego oraz opisano rolę pliku COMMAND.COM.

DODATKOWE POLECENIA WERSJI 3.0

Wprowadzenie wersji 3.0 systemu PC-DOS jest związane z pojawieniem się modelu AT oraz dołączonych do niego napędów dyskietek o poczwórnej gęstości zapisu (ang. high-capacity diskette drive). Istnieje, co prawda, jeszcze nowsza wersja 3.1, dostosowana do pracy w sieci i wzbogacona o polecenia współdzielenia plików, jednak nie będzie tu omówiona.

W wersji 3.0 wprowadzono kilka nowych poleceń (ATTRIB, LABEL, SELECT, SHARE) oraz rozszerzono semantykę kilku innych, dostosowując je do rozszerzeń sprzętowych (FORMAT, BACKUP, RESTORE, GRAPHICS, DATE).

ATTRIB [+/—R] file

Celem polecenia zewnętrznego ATTRIB jest zapewnienie możliwości ochrony plików przed niepożądanym zapisem. Jest to cecha, której brakowało poprzednim wersjom systemu PC-DOS. Plikowi o wyspecyfikowanej nazwie (w tym przypadku — file), nadaje się (+R) lub odejmuje (—R) atrybut R (ang. read-only, tylko odczyt). Przykładowo, polecenie:

```
C>ATTRIB +R AUTOEXEC.BAT
```

spowoduje nadanie plikowi AUTOEXEC.BAT atrybutu R, co można sprawdzić kolejnym poleceniem

```
C>ATTRIB AUTOEXEC.BAT
```

uzyskując odpowiedź

R C: AUTOEXEC.BAT

Jakikolwiek dostęp do takiego pliku z zamiarem zmiany jego zawartości jest niemożliwy, co jest sygnalizowane odpowiednim komunikatem:

C>DEL AUTOEXEC.BAT
Access denied

LABEL [d:] [etykieta]

Polecenie zewnętrzne LABEL służy do nadawania woluminom dyskowym etykiet, zazwyczaj nadawanych poleceniem FORMAT z parametrem /V. Jest możliwa zmiana istniejącej etykiety dysku, całkowite jej usunięcie lub przypisanie nowej etykiety. Etykieta może mieć długość najwyżej jedenastu znaków. Jeżeli zostanie wprowadzony dłuższy napis, to etykieta otrzyma tylko 11 pierwszych znaków. Etykietę można nadawać bezpośrednio poleceniem LABEL, np.:

A>LABEL B:JOLA

lub po odpowiedzi systemu operacyjnego, np.:

A>LABEL B:

Volume label (11 characters, ENTER for none)? JOLA

W obu przypadkach, sprawdzając czy dyskietce umieszczonej w napędzie B: została nadana etykieta JOLA

A>DIR B:

otrzymamy komunikat:

Volume in drive B is JOLA
Directory of B:

SELECT kraj klawiatura

Celem polecenia zewnętrznego SELECT jest jednokrotne utworzenie odpowiednich zawartości plików CONFIG.SYS i AUTOEXEC.BAT, zależnie od wartości wyspecyfikowanych parametrów: kraj i klawiatura (001 — USA, 044 — Wielka Brytania itd.). Na dyskietce źródłowej muszą istnieć programy przypisywania odpowiedniego znaczenia klawiszom klawiatury (KEYBxx) oraz programy DISKCOPY i DISKCOMP. Dla krajowych użytkowników polecenie to może być istotne przy dokonywaniu wyboru odpowiednich podzbiorów systemu.

SHARE [/F:obszar] [/L:locks]

Polecenie zewnętrzne SHARE służy do ładowania opcjonalnej części systemu operacyjnego, zapewniającej współdzielenie plików przez różne programy. Parametry F i L umożliwiają dostosowanie używanego obszaru pamięci do wymagań odpowiednich zastosowań. Parametr F określa liczbę bajtów obszaru przeznaczonego do rejestrowania informacji niezbędnej do współdzielenia plików (domyślnie 2048 bajtów), a parametr L służy do przydzielania obszaru niezbędnego dla określonej liczby tzw. blokad (ang. locks — wartość domyślna wynosi 20). Po wykonaniu polecenia SHARE, które powoduje załadowanie kodu kontroli współdzielenia plików, każde odwołanie do pliku (odczyt lub zapis) podlega weryfikacji pod względem zasad dostępu do rekordów. Nietrudno się domyślić, że polecenie to jest kosztowne w sensie zajętości pamięci — dla domyślnych wartości parametrów zajmuje ok. 5 KB pamięci operacyjnej.

ROZSZERZONE POLECENIA WERSJI 2.0

W związku z wprowadzeniem ulepszonego sprzętu, niektóre istniejące polecenia systemu PC-DOS rozszerzono o dodatkowe możliwości:

- polecenie FORMAT otrzymało nowy parametr /4 umożliwiający formatowanie dyskietek o poczwórnej gęstości
- polecenie BACKUP umożliwia składowanie plików z dowolnego dysku na dowolny inny (a nie tylko z dysku stałego na dyskietkę, jak dla wersji 2.0); odtwarzanie zeskładowanych plików jest możliwe tylko poleceniem RESTORE
- polecenie GRAPHICS umożliwia wybór rodzaju drukarki służącej do wydrukowania zawartości ekranu monitora przy użyciu klawisza PRTSC (COLOR1, COLOR4, COLOR8, COMPACT i GRAPHICS); dodatkowy parametr /R oznacza inwersję kolorów (biały zamiast czarnego i odwrotnie), a parametr /B — kolorowe tło
- dodatkowo, polecenie DATE umożliwia wprowadzanie daty w różnym formacie (mm-dd-yy, dd-mm-yy lub yy-mm-dd); polecenie to traci swoje znaczenie w modelu AT, który ma wbudowany bateryjny zegar czasu astronomicznego.

KONFIGUROWANIE SYSTEMU

System operacyjny PC-DOS ma pewne, choć dość ograniczone mechanizmy umożliwiające tworzenie jego konfiguracji zależnie od potrzeb użytkowników. Oprócz wymienionego w poprzednim odcinku artykułu pliku AUTOEXEC.BAT, który jest wykonywany zaraz po załadowaniu systemu operacyjnego, główną rolę w doborze i ustaleniu konfiguracji odgrywa plik CONFIG.SYS, nazywany plikiem konfiguracyjnym.

Jest to zwykły plik tekstowy, zawierający szereg poleceń w formie przypisania wartości odpowiednim zmiennym, przeszukiwany podczas ładowania systemu operacyjnego. Każdy rodzaj przypisania może wystąpić w pliku konfiguracyjnym tylko jeden raz (za wyjątkiem DEVICE). Poniżej opisano funkcje poszczególnych zmiennych (niektóre z nich występują tylko w wersji 3-COUNTRY, FCBS, LASTDRIVE).

BREAK = [ON|OFF]

Zmienna BREAK służy do ustalenia omówionej w drugim odcinku tego artykułu zasady przerywania działania programu klawiszem CTRL-BREAK. Mówiąc poglądowo, przypisanie:

BREAK = ON

umożliwia łatwiejsze i szybsze przerywanie wykonywania programów.

BUFFERS = n

Zmienna BUFFERS służy do określenia liczby buforów przeznaczonych do wymiany informacji między programami a plikami dyskowymi i może przybierać wartości od 1 do 99. Bufor jest obszarem pamięci przeznaczonym do przekazywania danych w czasie ich wymiany z dyskiem, jeśli ich długość nie jest równa rozmiarowi sektora. Im większa jest liczba buforów, tym więcej danych może być w pamięci, jednak wybór właściwej wartości n nie jest sprawą prostą. Jeżeli wykonywane są programy wymagające częstego dostępu do dysku, to ich działanie będzie dość wolne, gdy wartość n nie będzie dostatecznie duża. Z kolei, jeśli n będzie zbyt duże, to może zabraknąć pamięci dla wszystkich programów, gdyż każdy bufor przydziela jej ok. 0,5 KB. Typowa wartość n dla przetwarzania tekstowego może wynosić 2 lub 3, a dla systemu bazy danych 6—10.

COUNTRY = kraj

Zmienna COUNTRY przybiera wartości liczbowe służące do modyfikowania niektórych parametrów systemowych, zależnych od zwyczajów przyjętych w odpowiednich krajach. Dotyczy to takich elementów, jak: sposób zapisu daty i czasu dziennego, symbol jednostki monetarnej, sposób oddzielenia części dziesiętnej w zapisie liczb itp. Ciekawostką jest fakt, że wartości określające kraj są kierunkowymi numerami telefonicznymi odpowiednich krajów w komunikacji międzynarodowej, np. USA — 001, Wielka Brytania — 044, RFN — 049, Izrael — 972 itd.

DEVICE = file

Zmienna DEVICE umożliwia instalowanie tzw. ładowalnych programów obsługi urządzeń (ang. loadable device drivers), a więc nie wbudowanych do systemu operacyjnego, które są konieczne w przypadku niestandardowego wyposażenia komputera. Napis file musi określać nazwę programu obsługi i miejsce jego przechowywania, np. dla — dostarczanego wraz z wersją 3.0 systemu PC-DOS — programu obsługi dysku wirtualnego polecenie:

DEVICE = VDISK.SYS

oznacza zainstalowanie w pamięci programu znajdującego się na dysku domyślnym w skorowidzu pierwotnym (w ogólnym wypadku nie jest konieczne podawanie rozszerzenia SYS — akceptowane jest dowolne inne). Dysk wirtualny, nazywany też dyskiem elektronicznym, symuluje działanie zwykłego napędu dysku magnetycznego opartego na pamięci operacyjnej (minimalna pojemność pamięci operacyjnej, niezbędna do korzystania z tej możliwości, wynosi 192 KB), dzięki czemu zapewnia się szybszy dostęp do pamięci masowej, lecz bez permanentnego przechowywania informacji. Instalując dysk wirtualny, można mu nadać kilka parametrów przez przypisanie ich zmiennej DEVICE:

bbb — wielkość obszaru w kilobajtach (od 1 KB do maksymalnej pojemności pamięci, wartość domyślna — 64 KB)

sss — rozmiar sektora w bajtach (128, 256 lub 512, wartość domyślna — 128)

ddd — liczba elementów w skorowidzu (od 2 do 512, wartość domyślna — 64)

E — rozszerzenie pamięci powyżej 1 MB (parametr ten jest ważny tylko dla modelu AT z rozszerzoną pamięcią).

FCBS m,n

Zmienna FCBS (ang. file control blocks) jest wykorzystywana tylko w przypadku używania polecenia SHARE, tj. przy współdzieleniu plików. Wartość m określa dopuszczalną liczbę plików otwartych jednocześnie, a wartość n — maksymalną liczbę plików otwartych, chronionych przed automatycznym zamknięciem przez system operacyjny w przypadku przekroczenia dopuszczalnej liczby plików otwartych. Dopuszczalny zakres wartości m i n wynosi odpowiednio [1,255] i [0,255], a wartość domyślna — 4 i 0. Wartość m musi być większa lub równa n. Tę metodę określania dopuszczalnej liczby plików jednocześnie używanych stosuje się tylko w starej metodzie dostępu do plików, nie wymagającej przydzielania obszaru roboczego w samym systemie operacyjnym.

FILES = n

Zmienna FILES służy do określania dopuszczalnej liczby plików, do których istnieje jednoczesny dostęp przy użyciu metody wymagającej przydzielania obszaru roboczego wewnątrz systemu operacyjnego. Wartość n musi mieścić się w zakresie [8,255] — dla wersji 3 systemu operacyjnego (wartość domyślna n=8). Każde zwiększenie wartości n powyżej wartości domyślnej powoduje zwiększenie zajętego obszaru pamięci operacyjnej o 40 B.

LASTDRIVE = x

Zmienna LASTDRIVE określa maksymalną liczbę napędów dyskowych, które mogą współpracować z systemem PC-DOS. Wartością x jest leksykograficznie ostatnia litera alfabetu określająca odpowiedni napęd (wartość domyślna jest równa E). Nie można podać mniejszej liczby napędów niż jest rzeczywiście zainstalowane w systemie.

SHELL = file path switch

Zmienna SHELL służy do określenia nazwy kilku (file) zawierającego alternatywny interpreter poleceń, używany zamiast standardowego interpretera COMMAND.COM, jeśli występuje taka konieczność. Choć taka potrzeba zdarza się dość rzadko, głównie w wypadku opracowywania nowego oprogramowania podstawowego lub aplikacyjnego, zmienna SHELL jest często wykorzystywana do określenia miejsca przechowywania standardowego interpretera COMMAND.COM, jeżeli nie jest nim skorowidz pierwotny, np.:
SHELL = A:|JANUSZ|COMMAND.COM A:|JANUSZ| /P
Parametr P oznacza, że interpreter COMMAND-COM służy do rozpoczęcia pracy (ang. startup). Pominięcie tego parametru spowoduje niewykonanie poleceń zapisanych na pliku AUTOEXEC.BAT.

Nie są to wszystkie zmienne, których można używać w pliku CONFIG.SYS, ale tylko te są opisane w podręcznikach firmowych i dlatego nie zaleca się używania innych.

INTERPRETER POLECEŃ

Standardowy interpreter poleceń COMMAND.COM, zawierający m.in. kod poleceń, tablicę ich nazw, treść komunikatów systemowych itp., składa się z kilku oddzielnych części. Pierwsza z nich rezyduje na stałe w pamięci operacyjnej powyżej pliku IBMDOS.COM i służy do kontroli ładowania pozostałych części, które mogą być nakładkowane przez programy użytkowe. Druga część interpretera, tzw. kod inicjalizacji, służy do określenia granic pamięci dostępnej dla programów i do wykonania poleceń zapisanych na pliku AUTOEXEC.BAT. Ta część jest używana tylko do rozpoczęcia pracy i usuwana po wykorzystaniu. Trzecia, główna część interpretera, tzw. tymczasowa, rezydująca na końcu pamięci operacyjnej, zawiera kod wszystkich poleceń wbudowanych, jak DIR lub COPY. Mogą na nią zachodzić wykonywane programy użytkowe, co wymaga jej ponownego ładowania. Ostatnią część interpretera stanowi krótki fragment kodu przeznaczonego do ładowania poleceń zewnętrznych.

Ponieważ część pamięci zajętej przez interpreter poleceń może być zapisana przez dłuższe programy użytkowe, przy kolejnej próbie wykonania polecenia, którego kod w pamięci operacyjnej został zniszczony, PC-DOS podejmuje próbę ponownego ładowania interpretera. Dlatego zaleca się, aby domyślna jednostka dyskowa zawierała kopię tego interpretera. W przeciwnym wypadku, na ekranie monitora pojawi się komunikat:

Insert COMMAND.COM disk in A:
and strike any key when ready...

W celu sprawdzenia, czy tymczasowa część interpretera jest przechowywana w pamięci operacyjnej, część rezydentna wykonuje obliczenie sumy kontrolnej zawartości obszaru przechowującego część tymczasową. Jeżeli wartość sumy kontrolnej jest niepoprawna, to następuje ponowne załadowanie interpretera.

Jednym z głównych powodów wydzielenia interpretera poleceń w postaci samodzielnego pliku COMMAND.COM jest zapewnienie możliwości jego wymiany. Przykładowo, pierwsza wersja znanego procesora tekstowego EasyWriter ma swój własny interpreter poleceń.

Warto także zwrócić uwagę, że dzięki wydzieleniu interpretera poleceń jako pliku COMMAND.COM nie zawsze konieczne jest użycie klawiszy CTRL-ALT-DEL w celu ponownego zainicjowania systemu operacyjnego. Często wystarczy po prostu użycie polecenia COMMAND, wskutek czego nastąpi zainicjowanie systemu operacyjnego, polegające na załadowaniu interpretera poleceń i wykonaniu poleceń zapisanych na pliku AUTOEXEC.BAT, lecz bez ponownego ładowania plików IBMIO.COM i IBMDOS.COM.

* * *

W następnym, ostatnim odcinku opisu systemu PC-DOS dokończymy omawianie jego elementów składowych, opisując typowy przebieg procesu opracowywania programów.

WARUNKI PRENUMERATY NA 1987 R.

Prenumeratorzy zbiorowi — jednostki gospodarki uspołecznionej, instytucje i organizacje społeczne zamawiają prenumeratę dokonując wpłat na blankiecie „polecenie przelewu".

Prenumeratorzy indywidualni — osoby fizyczne zamawiają prenumeratę dokonując wpłaty w UPT lub NBP na blankiecie Wydawnictwa lub blankiecie NBP. Na odwrocie wszystkich odcinków blankietu należy wpisać tytuł czasopisma, okres prenumeraty, liczbę zamawianych egzemplarzy oraz wartość wpłaty.

Wpłacać należy na konto NBP III O/M Warszawa 1036-7490-139-11.

Prenumerata ulgowa — przysługuje wyłącznie osobom fizycznym — członkom SNT, studentom i uczniom szkół zawodowych. Warunkiem prenumeraty ulgowej jest poświadczenie blankietu wpłaty (przed jej dokonaniem) na wszystkich odcinkach pieczęcią Koła SNT, wyższej uczelni lub szkoły.

Sposób zamawiania prenumeraty taki sam jak dla prenumeraty indywidualnej.

Prenumerata ze zleceniem wysyłki za granicę — zamawia się tak jak prenumeratę indywidualną. Dodatkowo należy podać na blankiecie wpłaty nazwisko i dokładny adres odbiorcy. Cena prenumeraty ze zleceniem wysyłki za granicę jest dwukrotnie wyższa.

Przedpłaty na prenumeratę przyjmowane są w terminach:
— do 10 listopada na I kwartał, I półrocze i cały rok następny,
— do 28 lutego na II, III, IV kwartał i II półrocze,
— do 31 maja na III, IV kwartał i II półrocze,
— do 31 sierpnia na IV kwartał.

Uwaga!

Wpłaty na dwumiesięczniki przyjmowane są na okresy półroczne lub roczne.

Informacji o prenumeracie udziela — Zakład Kolportażu Wydawnictwa NOT-SIGMA, ul. Bartycka 20, 00-716 Warszawa, lub skr. poczt. 1004, 00-950 Warszawa, tel. 40-00-21 w. 249, 293, 297, 299 oraz 40-35-89 i 40-30-86.

Egzemplarze archiwalne czasopism — można nabyć za gotówkę w Klubie Prasy Technicznej w Warszawie, ul. Mazowiecka 12, tel. 27-43-65 lub zamówić w Dziale Handlowym Wydawnictwa, ul. Bartycka 20, skr. poczt. 1004, 00-950 Warszawa, tel. 40-37-31, na rachunek dla instytucji lub za zaliczeniem pocztowym dla osób fizycznych.

Cena miesięcznika INFORMATYKA została ustalona na 150 zł za numer (50 zł — cena ulgowa).

Cena prenumeraty wg cennika					
kwartalna		półroczna		roczna	
normalna	ulgowa	normalna	ulgowa	normalna	ulgowa
450	150	900	300	1800	600

Reaktywując w INFORMATYCE (nr 4/1985) dział Dydaktyki, redakcja udostępniła łamy czasopisma dla prezentacji nowych metod nauczania informatyki w naszych szkołach wyższych. Uważaliśmy — i dalej sądzimy — że o jakości i nowoczesności informatyki w Polsce zadecyduje przede wszystkim sposób jej nauczania. Nie jest przy tym istotny fakt wielu ograniczeń sprzętowych i finansowych, choć niewątpliwie znacznie utrudniają one to zadanie.

W kolejnych numerach zaprezentowaliśmy wyniki Seminarium Metodycznego Informatyki — Gdańsk, 22—26 września 1984 r. (INFORMATYKA, nr 4'1985), przedstawiliśmy propozycję wprowadzenia odrębnego kierunku nauczania w dziedzinie mikroinformatyki (INFORMATYKA, nr 5'1985) oraz zaprezentowaliśmy interesujące pomoce dydaktyczne usprawniające proces nauczania (INFORMATYKA, nr 8—9'1985). Wzięliśmy też udział w ogólnonarodowej dyskusji na temat wprowadzenia nauczania informatyki do szkolnictwa podstawowego i średniego (INFORMATYKA, nr 6—8'1985). W sumie jednak, trudno jest uznać dotychczasowy materiał za reprezentatywny przegląd dorobku uczelni w tej dziedzinie. Trudno również jest nam uwierzyć, że w wielu instytutach uczelni zajmujących się kształceniem informatyków nie ma interesujących rozwiązań w metodyce nauczania informatyki. Źle jest jednak, gdy rozwiązania te nie są udostępniane i wielu innych musi odkrywać już przetarte szlaki. Nie istnieje praktycznie wymiana między uczelniami elementów oprogramowania i sprzętu wspomagającego proces nauczania. Dlatego też w dalszym ciągu namawiamy pracowników uczelni do przedstawiania swoich koncepcji i rozwiązań na łamach INFORMATYKI. Na razie jednak, oczekując na materiały do publikacji, prezentujemy opracowanie zawierające propozycje programu studiów informatycznych dla małych uczelni amerykańskich. Niektóre z zamieszczonych w nim sformułowań wydają się oczywiste, ale warto je przypomnieć. Proponując Czytelnikom ten tekst, liczymy na zainteresowanie i współpracę. Oczekujemy na prezentację nowych konspektów zajęć, propozycji programów i przedmiotów. Zamieścimy także nadesłane streszczenia najciekawszych prac doktorskich i magisterskich, opisy pomocy dydaktycznych dla wspomagania nauczania informatyki itp. (W.I.)

Program nauczania informatyki w małych uczelniach amerykańskich

Lawinowo wzrastające zapotrzebowanie na informatyków przy ograniczonych możliwościach nauczania, spowodowało potrzebę wprowadzenia kierunków informatycznych (ang. computer science) do wielu szkół inżynierskich (ang. colleges) oraz małych lokalnych uniwersytetów. Jednocześnie szkół tych, mających wiele różnych profilów nauczania, nie stać na opracowanie i pełne wdrożenie programów nauczania z doświadczonych dużych uczelni. Dlatego też powstał problem opracowania nowych, nowoczesnych programów nauczania informatyki, opartych na dotychczas istniejących programach, takich jak: Curriculum'78 „Information Systems Curriculum Recomendations for the 80's, opracowanych przez ACM lub „Modern curriculum", opracowanych przez IEEE. Programy te zostały jednak opracowane przy założeniu wprowadzenia odrębnego, pełnego profilu wykształcenia w zakresie informatyki.

Grupa specjalistów z komitetu edukacyjnego ACM opracowała więc program nauczania informatyki dla małych uczelni amerykańskich, przyjmując następujące założenia:

- część kursów jest wspólna dla wszystkich profili kształcenia występujących w danej szkole
- część kursów jest obieralna, z możliwością wyboru różnych opcji i zestawów
- uwzględnia się cele i funkcje szkoły
- uwzględnia się wielkość szkoły oraz możliwości jej zespołu nauczającego, zasoby finansowe i dostęp do sprzętu komputerowego.

Zorganizowanie studiów jest przedsięwzięciem kosztownym i wymaga zapewnienia stałego dopływu funduszy. Rozmiary i kierunki szkolenia muszą być szczególnie w małych szkołach dobrze planowane i kontrolowane. W przeciwnym razie koszty zorganizowania szkolenia mogą szybko przekroczyć dostępne fundusze. Zasoby niezbędne dla kształcenia można podzielić na trzy kategorie:

- wysoko kwalifikowana kadra nauczycielska
- sprzęt komputerowy wraz z oprogramowaniem
- bogato zaopatrzona biblioteka książek, czasopism i dokumentacji oprogramowania.

PODSTAWY FUNKCJONOWANIA SZKOŁY

Liczba potrzebnych nauczycieli akademickich zależy od liczby studentów i liczby kierunków specjalizacji. Przy organizowaniu studiów inżynierskich wymaganych jest co najmniej dziesięć kursów-przedmiotów oraz jeden przedmiot wprowadzający.

Zgromadzenie odpowiedniej liczby nauczycieli ze stopniem doktorskim i dużym doświadczeniem jest bardzo trudne. Kadry takiej jest mało, a ponadto przemysł przedstawia bardziej atrakcyjne oferty pracy. W konsekwencji szkoła musi zatrudnić informatyków nie posiadających stopnia doktorskiego — ewentualnie wykorzystując specjalistów z innych dziedzin. Obie te możliwości wymagają weryfikowania umiejętności. Kwalifikowanie może odbywać się na podstawie publikacji i prowadzenia prac nadających się do opublikowania, a także na podstawie udziału w sympozjach i konferencjach, działalności w towarzystwach specjalistycznych oraz działalności konsultacyjnej.

Ważnym problemem jest **wybór typu sprzętu.** Powinny być dostępne co najmniej dwa różne systemy komputerowe. Nie oznacza to jednak, że na terenie uczelni musi znajdować się kilka typów komputerów.

Zakłada się, że każdy student powinien mieć **dostęp do komputera** co najmniej przez pięć godzin w tygodniu. Nie bez znaczenia jest wygląd miejsca pracy i wygoda użytkownika systemu (dobra widoczność klawiatury i ekranu, wygodny edytor i system operacyjny). Zalecane jest wykorzystanie kompilatorów języków tradycyjnych, takich jak Pascal, Fortran, Cobol oraz języków nowszych, jak Modula-2, Ada, a także języków specjalnego przeznaczenia, jak Snobol, Lisp i Prolog.

Zaopatrzenie biblioteki w czasopisma, książki i dokumentację przeznaczoną dla prowadzących zajęcia i studentów jest kosztowniejsze niż w innych dziedzinach. Konieczna jest bowiem stała aktualizacja księgozbioru, który w dziedzinie informatyki starzeje się bardzo szybko. Niezbędny jest stały dostęp do takich czasopism, jak: Communications of the ACM, Computing Surveys, Transactions of the IEEE, wszelkiego rodzaju Newsletters, Data Management, Computer oraz — wybranych czasopism specjalistycznych i popularnonaukowych. W toku kształcenia bardzo przydatne są pomoce audiowizualne, jak filmy, filmy animowane, wideokasety, a także prezentacje zastosowań informatyki w przemyśle.

PROGRAM NAUCZANIA

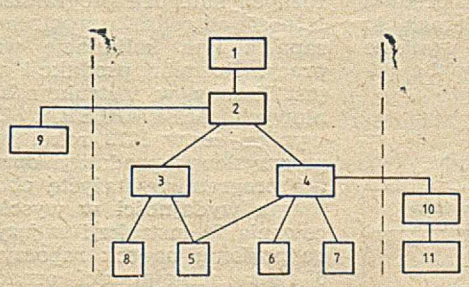
Poprawnie skonstruowany program powinien mieć wyraźnie określone cele nauczania (jeden lub kilka). Możliwe jest dążenie do realizacji różnych celów nauczania i istnieje wiele dróg dalszej kariery studentów. Pomijając różnice wynikające z lokalnej sytuacji szkoły można wyróżnić trzy ogólne zasady dotyczące organizatorów i realizatorów programu:

1. Istnieje wyraźne rozróżnienie pomiędzy tymi, którzy uzyskali stopnie naukowe w dziedzinie informatyki a tymi, którzy uzyskali takie stopnie w innych dziedzinach.
2. Opracowane zalecenia do programu kształcenia stanowią doskonałą pomoc, jednakże do skonstruowania poprawnego programu dla danej szkoły potrzebna jest intuicja i wiedza przekraczająca ramy opracowywanych zaleceń.
3. Kadra nauczycielska mająca wykształcenie w innych dziedzinach musi rozumieć dwie powyższe różnice.

Aby zilustrować przedstawione stwierdzenia, warto uświadomić sobie różnicę pomiędzy sposobem wykładania przedmiotu „Informatyka" — na przykład — dla studentów Wydziału Mechaniki Precyzyjnej, a przedmiotu „Podstawy informatyki" dla studentów Informatyki. W pierwszym wypadku uwypuklony jest problem zastosowania wybranego języka programowania, natomiast w drugim, język programowania jest stosowany jedynie jako narzędzie do wyrażania idei algorytmów.

Główne przedmioty programu nauczania

Zaprezentowany w artykule program nauczania uwzględnia te działy informatyki, które są najistotniejsze dla kształcenia inżynierów. Końcowy profil studiów zależy od doboru przedmiotów i od uwypuklenia poszczególnych zagadnień. Na rysunku przedstawiono wzajemną zależność proponowanych przedmiotów. Osiem przedmiotów stanowi trzon edukacji. Ponadto, uwzględniony jest jeden dodatkowy przedmiot, dotyczący filozofii języków programowania i dwa dodatkowe przedmioty z dziedziny nauki o systemach informacyjnych. Spis głównych przedmiotów przedstawiono poniżej.



Struktura kolejności przedmiotów programu edukacji

Przedmiot 1 i 2: Elementy programowania

Dwusemestralny kurs daje podstawy sztuki programowania. Studentom prezentuje się metodykę programowania oraz uczy się ich poprawnego stylu programowania według ogólnie przyjętych praktyk.

Przedmiot 3: Organizacja maszyny i programowanie w języku asemblera

Przedmiot ten uwypukla podstawy konstrukcji komputera i jego architektury. Konieczne jest wykorzystanie języka asemblera dostępnej maszyny oraz ilustrowanie wykładów pokazami sprzętu. Należy zwracać uwagę na to, aby nie skupiać zainteresowań na jednej konkretnej architekturze.

Przedmiot 4: Struktury i przetwarzanie plików

Przedmiot ten zaznajamia studentów z metodami organizacji struktur plików oraz z efektywnymi metodami dostępu do informacji w plikach. Studenci powinni praktycznie zapoznać się z plikami o organizacji sekwencyjnej, bezpośredniego dostępu, indeksowej oraz z przetwarzaniem i aktualizowaniem plików. Jednym z użytych narzędzi może być język Cobol.

Przedmiot 5: Systemy operacyjne

Kurs ten zapoznaje studentów z ideą konstrukcji systemów operacyjnych i sterowania tymi systemami. Prezentowane są konstrukcje znanych systemów operacyjnych i języków poleceń operatorskich dla tych systemów. Uwzględnione są także elementy programowania współbieżnego i implementacji systemu plików.

Przedmiot 6: Systemy baz danych

Prezentowana jest historia głównych rodzajów baz danych. Przewidywane są praktyczne ćwiczenia co najmniej z jednym systemem baz danych z operacjami wyszukiwania, aktualizacji oraz definiowania struktury bazy. Zaleca się uwypuklenie relacyjnego modelu bazy danych wraz z naturalnymi językami dostępu do bazy.

Przedmiot 7: Struktury danych i algorytmy

Przedmiot obejmuje przegląd typowych struktur danych oraz metod reprezentacji tych danych w pamięci operacyjnej i pomocniczej wraz z zasadami alokacji dynamicznej. Kurs powinien też uwzględniać metody pomiaru złożoności algorytmów.

Przedmiot 8: Architektura systemów

Jest to równoległy do przedmiotu „Systemy operacyjne" przedmiot traktujący o analizie i projektowaniu systemów operacyjnych wraz z praktycznymi aspektami konstruowania systemów oprogramowania.

Przedmiot 9: Języki programowania

Przedmiot obejmuje porównanie języków programowania pod względem ich właściwości. Przy braku odrębnego przedmiotu dotyczącego translatorów, przedmiot ten powinien obejmować elementy metod translacji.

Przedmiot 10: Koncepcja systemów informacyjnych

W ramach przedmiotu prezentowana jest idea systemów informacyjnych ze szczególnym uwypukleniem ich analizowania, projektowania i dokumentowania. Uwzględnione są problemy zarządzania systemami informacyjnymi, zespołami programistów oraz ukierunkowaniem języków programowania na definiowanie struktur informacji.

Przedmiot 11: Projektowanie, implementowanie i pielęgnowanie systemów oprogramowania

Przedmiot dotyczy metod tworzenia i pielęgnowania kompleksowych systemów cyfrowych złożonych ze sprzętu i oprogramowania. Uwzględnia się ćwiczenia grupowe oraz przedstawia problemy tworzenia i użytkowania oprogramowania narzędziowego.

* * *

Prezentowany program nauczania przedstawiono w sposób bardzo skrócony. Brak jest w nim kursu matematyki i praktycznie pominięto tematykę sprzętu. Ta minimalna wersja programu może być rozbudowana z uwypukleniem zastosowań praktycznych (zarządzanie, przemysł) lub z podkreśleniem teorii programowania. Program ten może być również zastosowany do kształcenia informatyków specjalizujących się w pracach rutynowych.

Opracował: PAWEŁ GRZEGORZEWICZ
na podstawie Communications of the ACM, Vol. 28, pp. 605—611, June 1985

Programista zawodowy i wartość jego pracy

Pracownia Zastosowań Informatyki (PZI) Instytutu Informatyki Uniwersytetu Wrocławskiego (IIUWr.) skupia pracowników, których kwalifikacje i zadania pozwalają określać ich jako programistów zawodowych. PZI nie wytwarza oprogramowania metodami przemysłowymi. Zważywszy jednak prawidłowość, że produkt wytwarzany metodami przemysłowymi jest (winien być) tańszy od produktu wykonanego metodami nieprzemysłowymi, porównanie norm wydajności dla programistów i cen globalnych produktów programowych, wytwarzanych na skalę przemysłową może być przyczynkiem do szacowania pracy i wydajności komórki nieprzemysłowej, jaką jest PZI IIUWr.

Aczkolwiek PZI nie produkuje metodami przemysłowymi, to jednak zatrudnieni w tej pracowni programiści (jak i ogół programistów IIUWr.) wytwarzają oprogramowanie, które od czasu do czasu trzeba wycenić i sprzedać. Warto też znać przeliczeniową wartość produktów programowych wytwarzanych na potrzeby IIUWr lub uczelni, a więc nie podlegających sprzedaży.

Sytuacja taka inspiruje do tworzenia rozwiązań w dziedzinie cen i kalkulacji oprogramowania, a także do rozeznania tego rodzaju rozwiązań w otoczeniu.

Poniżej przytoczono wyciąg informacji z dokumentu [1], określającego normy i ceny dla przemysłowej produkcji oprogramowania. Należy zaznaczyć, że są to dane z 1973 roku. Przeglądając zawarte w dokumencie tabele trzeba wziąć nieodzowne poprawki na postęp dokonany w informatyce, w szczególności w metodach wytwarzania oprogramowania, jak też uwzględnić czynnik inflacji.

Autor sądzi, że z uwagi na wspomniany postęp, obecne ceny i normy wydajności kształtują się korzystniej — niestety, nie dysponuje danymi aktualnymi.

Celowo ograniczono się do zacytowania tabel cenowo-wydajnościowych, ponieważ inne dane w dokumencie źródłowym, na przykład nakład pracy na poszczególne etapy wytwarzania oprogramowania czy definicje produktu programowego jako sumy komponentów, dają się łatwo uzyskać z literatury bieżącej.

W tabelach 1 i 2 przedstawiono dane dotyczące względnej wartości instrukcji w niektórych językach oraz „ceny zbytu" pojedynczych instrukcji. Z tabeli 1 wynika, że jedna instrukcja w Fortranie równa się czterem instrukcjom przeliczeniowym w języku PLAN. Cena w tabeli 2 odnosi się do instrukcji przeliczeniowej i uwzględnia: projektowanie ogólne, testowanie projektu, projektowanie szczegółowe, kodowanie, uruchamianie, scalanie i dokumentowanie oraz testowanie produktu — 65% nakładu pracy według założeń z 1973 roku. Cena nie uwzględnia rozpoznania wstępnego i opracowania wymagań oraz przygotowania do rozpowszechniania i pielęgnacji (według danych z 1973 roku, odpowiednio: 10%, 5%, 20% nakładu pracy).

Tabela 1. Współczynniki przeliczeniowe instrukcji w zależności od rodzaju języka programowania

Języki	Plan	Assembler	Fortran	Cobol
Plan	1	0,7	0,25	0,3
Assembler	1,4	1	0,3	0,4
Fortran	4	3,33	1	—
Cobol	3	2,5	—	1

Przez komponent rozumie się program (grupę podprogramów) o taryfikowanym stopniu złożoności. Dokument [1] nie rozstrzyga metod klasyfikacji pod tym względem.

Tabela 2. Cena zbytu jednej instrukcji

Typ komponentu	Cena
Prosty	460 zł — 510 zł
Średnio złożony	690 zł — 770 zł
Złożony	920 zł — 1030 zł
Bardzo złożony	1380 zł — 1540 zł
Specjalny	1960 zł — 2200 zł

Rozważmy przykład, w którym danych jest 6 komponentów prostych o łącznej liczbie instrukcji 2300, 5 komponentów złożonych — zawierających 1600 instrukcji i 1 komponent bardzo złożony, mający 380 instrukcji. Cena za wykonanie etapów, które obejmuje tabela, wyniesie 1 584 400 zł. Pełen cykl wytwarzania produktu będzie kosztował 2 440 600 zł. Jest to cena zbytu dla generalnego dostawcy lub produkcji jednostkowej. Jeśli produkt będzie użyty na 100 komputerach, to cena zbytu dla użytkownika wyniesie 25 200 zł, z wliczeniem opłaty dla generalnego dostawcy.

Tabela 3. Kategorie programistów zawodowych

Nazwa i kategoria	Stanowisko
Programista zawodowy kat. V	Główny analityk branżowy
Programista zawodowy kat. IV	Starszy analityk (starszy projektant)
Programista zawodowy kat. III	Analityk
Programista zawodowy kat. II	Programista
Programista zawodowy kat. I	Młodszy programista
Programista-stażysta	Stażysta

Tabela 4. Roczna wydajność pracy programisty kategorii III

Typ komponentu	Rozmiar produkcji (liczba instrukcji)
Prosty	3000
Średnio złożony	2000
Złożony	1500
Bardzo złożony	1000
Specjalny	700

Tabela 5. Przeliczeniowa wydajność pracy programisty zawodowego w relacji do programisty zawodowego kategorii III

Kategoria	Współczynnik
V	1,9
IV	1,5
III	1,0
II	0,6
I	0,3
Stażysta	0,1

Tabela 6. Sposób kalkulacji wartości rocznego produktu pracy programisty zawodowego kategorii III

Składnik kosztów	Udział w tys. zł/rok
Płace, ubezpieczenia itp.	120
Renta umiejętności	400
Amoratyzacja maszyn i urządzeń	500
Materiały, drobne zakupy, usługi zewnętrzne	50
Narzut kosztów ogólnych	135
Razem koszt	1250
Zysk 10%..24%	...
Cena zbytu	**1375..1538**

Tabele 3—6 zawierają dane dotyczące zróżnicowania wydajności pracy różnych kategorii programistów, w zależności od rodzaju programów, i związaną z tym kalkulację wartości tej pracy. Rozmiar produkcji w tabeli 4 jest liczony w instrukcjach przeliczeniowych ASSEMBLERA. Z tabeli 5 wynika, że w 1973 roku, w zakładzie przemysłowej produkcji oprogramowania programistę-stażystę obowiązywała norma wydajności w wysokości opracowania 70 instrukcji w języku niskiego poziomu na rok roboczy. W tabeli 6 renta umiejętności oznacza wpłatę na rzecz odtworzenia funduszu WOG, na prace rozwojowe itp. Amortyzację oblicza się przy założeniu czteroletniego okresu amortyzacji środków technicznych. Według tej kalkulacji programista zawodowy III kategorii daje roczny przerób złotówkowy na sumę 1 375 000—1 538 000 zł.

Według własnych wyrywkowych szacowań nakład czasu na instrukcję kodu wewnętrznego w cyklu wytwarzania oprogramowania — rozpatrywanym od założeń wstępnych (ale bez dyskusji celów) aż po przetestowany i udokumentowany wyrób — możemy określić na 1/30 godziny intensywnej pracy programisty. Podobnie, dla jednej instrukcji języka opisu zadań GEORGE 3, czas ten wynosić może $^1/_{10}$ godziny.

Podane liczby należy interpretować z ostrożnością, ponieważ:
- oszacowanie dotyczy krótkich, jednostkowych zadań, w których nie zachodziła potrzeba uzgodnień, dyskusji i koordynacji (można przyjąć, że konieczna kooperacja wydłuża czas trzykrotnie)
- należy zastosować współczynnik lenistwa (odwrotność intensywności pracy) rzędu nawet 200%
- wyniki pochodzą z jednostkowych eksperymentów wykonanych przez jednego programistę.

Tak więc sygnalizowana wydajność może w szerszym kontekście wynieść 30:3:3≈3 instrukcje kodu na godzinę.

Z tabeli wydajności pracy programisty kategorii III otrzymujemy dla średniej trudności zadania programistycznego wydajność roczną równą 1500 instrukcji przeliczeniowych ASSEMBLERA. Przyjmując umownie 35-godzinny tydzień pracy i 47-tygodniowy rok roboczy, otrzymujemy na tej podstawie godzinową wydajność: 1500:47:35≈1 instrukcja.

Wynik ten jest trzykrotnie niższy od pierwszego oszacowania wydajności pracy programisty. Co jest powodem otrzymanej dysproporcji? Istnieją trzy możliwości:
- obecnie programista zawodowy pracuje trzy razy wydajniej niż przed 10 laty
- oszacowanie czasochłonności zawarte w cenniku źródłowym było (jednak) zawyżone wobec realnych możliwości programistów
- przedstawiona kalkulacja jest zbyt optymistyczna i przecenia potencjał wytwórczy statystycznego programisty zawodowego.

Określmy średnią płacę programisty zawodowego na 14 000 zł miesięcznie. Przy miesiącu roboczym liczącym 170 godzin i około 40% wskaźniku kosztów ubezpieczenia, pierwszy element kalkulacji — według tabeli 6 — wyniesie 120 zł/godz. Godzina robocza programisty zawodowego stanowi w kalkulacji z tabeli 6 jedenastą do trzynastej części ogólnej wartości produktu jego pracy. Biorąc wielkość dwunastokrotną, jako ogólny koszt produktu pracy programisty, otrzymujemy wartość:
120×12=1440 zł/godz.
co w rozłożeniu na 3 instrukcje kodu daje:
1440:3=480 zł na instrukcję.
Liczba ta jest przeszło dwukrotnie mniejsza od ceny zbytu jednej instrukcji (dla złożonego komponentu) z tabeli 2. Dodatkowo należy wziąć pod uwagę relatywną wartość pieniądza obecnego i sprzed 10 lat.

Jak skomentować ten wynik? Jeśli odrzucić trzecią możliwość odpowiedzi na uprzednio postawione pytanie, to programiście zawodowemu pozostaje... czym prędzej złożyć u swojego chlebodawcy podanie o podwyżkę.

* * *

Należy sądzić, że podniesiona tu kwestia oszacowań wydajnościowo-cenowych pracy programisty zawodowego interesuje ogół zatrudnionych programistów i nie tylko ich. Jak widać, do sprawy kalkulacji wartości pracy i wydajności programisty można podchodzić różnorodnie. Nie trzeba uzasadniać, że wiedza o wartości pracy programisty zawodowego powinna być ogólnie dostępna, ale najpierw trzeba ją poznać.

Zapraszam do dyskusji [1]).

Autor dziękuje mgr. Zdzisławowi Jarzębowskiemu za pomoc w dotarciu do materiału źródłowego.

ZDZISŁAW PŁOSKI

[1]) Zainteresowanym czytelnikom przypominamy, że o cenniku usług informatycznych pisaliśmy w numerach 3—5/1980 (Red.).

LITERATURA

[1] Organizacja produkcji oprogramowania maszyn cyfrowych oraz metodyka i założenia tych cen produktów. Zakłady Przyrządów Automatyki i Aparatury Pomiarowej MERA. Warszawa, listopad 1973.

KONFERENCJE

EUROMICRO '87

W dniach od 14 do 17 września 1987 r. w Portmouth (W. Brytania) odbędzie się trzynasta doroczna konferencja EUROMICRO '87 pod hasłem *„Mikrokomputery: metody i konstrukcje"*. Organizatorem konferencji jest EUROMICRO — Europejskie Stowarzyszenie Mikroprzetwarzania (European Association for Microprocessing and Microprograming).

Program konferencji przewiduje dwa rodzaje obrad: sesje naukowe, przeznaczone do prezentacji osiągnięć w dziedzinie prac badawczo-rozwojowych oraz sesje komunikatów, przeznaczone do zwięzłych prezentacji praktycznych wyników prac, zwłaszcza w dziedzinie zastosowań.

Proponowany program konferencji jest niezwykle obszerny i obejmuje następującą tematykę:

- problematyka sieci,
- interfejsy i magistrale,
- mikrokomputerowe stanowiska pracy,
- mikrokomputery w aparaturze naukowej,
- zbieranie i weryfikacja danych,
- systemy wbudowane,
- komputerowa integracja wytwarzania (CIM),
- automatyzacja biur,
- zaawansowane zastosowania mikrokomputerów,
- przetwarzanie obrazów,
- robotyka,
- systemy operacyjne i powiązanie użytkownika z oprogramowaniem,
- języki czwartej generacji,
- inżynieria oprogramowania i problemy środowiska,
- rozwój sprzętu,
- języki opisu sprzętu,
- techniki specyfikacji,
- kompilacja układowa,
- analiza i wydajność systemów,
- systemy rozproszone i tolerancja błędów,
- przetwarzanie współbieżne,
- architektura i projektowanie systemów wieloprocesorowych,
- architektura RISC dla GaAs,
- architektura sztucznej inteligencji,
- systemy ekspertowe.

Szczegółowe informacje otrzymać można od przewodniczącego komitetu organizacyjnego konferencji: John Billingsley, Portsmouth Polytechnic Faculty of Engineering, Anglesea Rd. Portsmouth P01 2DJ (United Kingdom), tel. (44) (705) 82 76 81.

Poprawa efektywności testowania wymienności taśm magnetycznych w systemach ODRA 1300

Jednym z podstawowych kryteriów oceny sprawności technicznej pamięci taśmowych jest tzw. *wymienność taśm magnetycznych*. Właściwość ta, charakteryzująca sprawne technicznie i prawidłowo zestrojone urządzenia pamięci taśmowej, określa ich zdolność do odczytania informacji zapisanych w tym samym standardzie na innych pamięciach taśmowych i odwrotnie.

Osiągnięcie pełnej wymienności wszystkich urządzeń pamięci taśmowych jest w każdym ośrodku obliczeniowym jednym z podstawowych zadań służb technicznych. Taśma magnetyczna jest bowiem w chwili obecnej podstawowym nośnikiem oprogramowania i danych wprowadzanych do systemów komputerowych, zwłaszcza w sytuacji pełnego rozpowszechniania taśmowych systemów rejestracji danych typu MERA 9150 lub SEECHECK.

Podstawowym narzędziem służącym ocenie wymienności taśm magnetycznych, będącym w dyspozycji służb technicznych, są testy. Producent maszyn cyfrowych ODRA 1300 dostarcza dwa takie programy: ≠TMTA i ≠INTA. Testy te po uruchomieniu dokonują na każdym przewijaku pamięci taśmowej zapisu odpowiedniej liczby danych testowych w postaci bloków o zmiennej długości i zawartości, po uprzednim odczytaniu danych zapisanych w identyczny sposób na innych jednostkach pamięci. Taśmę przekłada się podczas pracy testu na kolejne urządzenia, testując w ten sposób poprawność ich działania.

Informacji o wyniku testowania dostarcza egzekutor, wyprowadzając na konsolę operatorską (po zwolnieniu taśmy przez program i jej rozładowaniu) informację o sumarycznej liczbie wykonanych powtórzeń odczytu lub zatrzymując program z komunikatem typu FAIL n w wypadku niemożliwości prawidłowego odczytania danych. Informacje te nie pozwalają jednak jednoznacznie określić, które z testowanych pamięci taśmowych są źródłem wykrytych przez test błędów. Nie wiadomo bowiem, jak rozkładają się na poszczególne przewijaki naliczone przez egzekutor powtórzenia odczytu. Pewne wnioski można wyciągnąć na podstawie uważnej obserwacji pulpitu technicznego jednostki sterującej pamięciami taśmowymi MTS-304. Jest to jednak czynność uciążliwa, a jej wynik nie jest jednoznaczny i łatwy do zanalizowania.

W celu poprawienia testowania wymienności taśm magnetycznych dokonano modyfikacji testu ≠TMTA polegającej na:
1) zastosowaniu własnego monitorowania przez program operacji odczytu bloku danych (tzw. tryb addytywny ≠30000), wskutek czego egzekutor dokonuje tylko jednej próby odczytu bloku danych, po czym przekazuje sterowanie programowi z informacją o ewentualnym błędzie odczytu umieszczoną w słowie odpowiedzi,
2) wykonaniu zliczania powtórzeń odczytu wewnątrz nagrania z danej pamięci taśmowej i wyprowadzania uzyskanych tą drogą informacji na drukarkę wierszową lub monitor operatorski (po sprawdzeniu zawartości słowa odpowiedzi program sam wykonuje ewentualne cofnięcie taśmy o jeden blok i powtórny odczyt).

Informacje uzyskane w wyniku przebiegu zmodyfikowanego w ten sposób testu pozwalają dokładnie określić, które urządzenia wymagają naprawy i strojenia oraz wskazać przyczynę ich nieprawidłowego działania (układy odczytu, zapisu lub sterowania napędem rolki). Wynik działania testu w postaci wydruku zawierającego datę, godzinę testowania oraz zestawienie liczby powtórzeń lub błędów jest odzwierciedleniem stanu technicznego zestawów pamięci taśmowych oraz bardzo przydatnym materiałem statystycznym pomocnym w prowadzeniu prac konserwacyjnych i remontowych.

KRZYSZTOF LISIAK

Centralny Ośrodek Informacji Górnictwa Oddział nr 2 Dąbrowa Górnicza

Wielodostępne systemy operacyjne dla IBM PC/XT (1)

Wielozadaniowymi i wielodostępnymi systemami operacyjnymi przeznaczonymi dla komputerów osobistych IBM PC/XT są PICK, COHERENT i THEOS. THEOS jest to nowa nazwa dla systemu operacyjnego Oasis, w wersji Oasis 86. Żaden z tych systemów nie akceptuje programów działających w systemie MS-DOS, tak więc nie można w nich używać znanych pakietów arkuszy elektronicznych, przetwarzanie tekstów i innych, dla których w większości klienci decydują się na kupno IBM PC/XT.

Wymienione systemy operacyjne są przeznaczone do bardziej specjalizowanych celów. PICK i THEOS zostały zaprojektowane jako podstawa dla wielodostępnych baz danych. COHERENT jest oparty na bardziej ogólnych zasadach (jest to wersja Unixa). Jego najbardziej odpowiednim zastosowaniem jest programowanie w języku C.

Ze względu na duże różnice pomiędzy poszczególnymi sytemami wynikające z faktu, iż każdy z nich ma służyć innemu celowi, nie ma sensu na porównywanie ich poszczególnych cech. Zamiast tego, każdy z nich zostanie oddzielnie scharakteryzowany i przeanalizowany. Sprawdzeniu poddano następujące wersje poszczególnych systemów: PICK 1.3, COHERENT 2,3,43 i THEOS 7,0. PICK i COHERENT (wraz z kompilatorami języków programowania) kosztują 495 dolarów każdy, a THEOS — 595 dolarów (bez kompilatorów). Dodatkowy koszt Basica lub języka C dla systemu THEOS wynosi 400 dolarów. Żaden z tych systemów nie ma zabezpieczenia przed skopiowaniem.

Wymienione systemy testowano na IBM PC/XT składającym się wyłącznie z urządzeń produkcji IBM (włącznie z pakietem pamięci). Konfiguracja mikrokomputera była następująca:
— stały dysk 10 MB,
— dysk elastyczny 360 KB,
— 512 KB pamięci RAM,
— monitor monochromatyczny.

Używano także terminala Zenith Z-19, pracującego z szybkością transmisji 9600 bodów, w celu wypróbowania pracy wieloużytkowej każdego z systemów operacyjnych. Drukarkę dołączono do wyjścia równoległego. Należy przypuszczać, że wszystkie systemy będą działały na różnych konfiguracjach sprzętowych, lecz autor nie próbował ich używać na odpowiednikach IBM

PC innych firm, jak również nie pracował z innymi typami dysków stałych.

Systemy te są zbyt złożone, aby można utrzymywać, że przedstawiono je dokładnie. Każdemu z nich poświęcono zaledwie parę dni, w tym czasie zdołano zbadać około 10% możliwości ich funkcji. Dzięki takiemu próbkowaniu ich niedociągnięcia stały się bardziej widoczne niż w wypadku pełnej znajomości.

PICK

PICK jest systemem operacyjnym zaprojektowanym do zadań specjalnych (por. INFORMATYKA, nr 5, 1985, str. 28) przy współpracy z bazą danych. Używając PICKA, użytkownik nie odczuwa granicy między systemem operacyjnym a bazą danych. Na przykład PICK nie posiada poleceń listowania nazw plików ze skorowidza. Zamiast nich używane są dyrektywy wspomagające bazę danych, dzięki czemu można otrzymać listing pozycji w słowniku konta użytkownika. Polecenia nazywane są tu czasownikami, rekordy — pozycjami, a pola — atrybutami. Dla kogoś, kto do tej pory pracował z systemami tradycyjnymi, PICK wydaje się dość dziwny, ale po dniu lub dwóch można do niego przywyknąć.

Najbardziej znamienną cechą PICKA jest to, iż każdej jego funkcji odpowiada jeden wyspecjalizowany program, a nie wiele programów. PICK jest wyposażony tylko w jeden edytor tekstowy, służący do edycji plików źródłowych, poleceń do zapamiętania, rekordów danych i wpisów słownikowych. Istnieje także jeden typ plików danych; jedna pozycja może być rekordem danych — wtedy pola mają swoje zwykłe znaczenie lub programem źródłowym albo dokumentem — wtedy pola odpowiadają poszczególnym liniom tekstu. Jednolite traktowanie wszelkich obiektów i prostota stanowią o ławości posługiwania się PICKIEM, gdy tylko zrozumie się mechanizm jego działania. Te cechy niewątpliwie przyczyniają się do sprawności i pewności działania PICKA.

Patrząc od wewnątrz, PICK traktuje komputer dwojako. Po pierwsze — dane i programy odwołują się do pamięci wirtualnej. System dzieli obszar pamięci fizycznej na 512-bajtowe ramki. Dane i części programów są w miarę potrzeby przesyłane z dysku do pamięci operacyjnej. Po drugie — programy są tłumaczone na język asemblera, który nie jest kodem 8088, ale kodem dla komputera wirtualnego, implementowanym przez interpreter głęboko w jądrze PICKA.

Taka interpretacja jest konieczna do pracy pamięci wirtualnej komputera PC/XT, gdyż nie ma on sprzętowego rozszerzenia pamięci. Interpreter jest tak skonstruowany, że zapewnia użytkownikowi wyłączność korzystania z przydzielonego mu obszaru pamięci. Stanowi to problem dla innych wielodostępnych systemów dla PC/XT, gdyż komputer nie ma sprzętowej ochrony dostępu do pamięci.

Baza danych ma strukturę hierarchiczną, przedstawioną w tabeli. Zainteresowanych strukturą plików PICKA odsyłamy do artykułu: „PICK Operating System. Part 1 — Information Managment" R. Cook, J. Bardon, BYTE, p. 176, October 1984. Elastyczność wpisów, atrybutów, wartości i podwartości pozwala na organizowanie danych w sposób znacznie odbiegający od innych systemów baz danych.

Rozpatrzymy przykładową bazę danych dla potrzeb szpitala, bądź przychodni. Podstawowym rekordem są tu dane o pacjentach, a dodatkowymi — rekordy opisujące każdą z wizyt. Możliwe jest również utrzymanie danych w postaci pojedynczego wpisu, który może posiadać atrybut nazwany np. wizyta. Dzięki takiej organizacji bardzo łatwo jest posługiwać się programem PICKA, gdyż wpis jest czytany i zapisywany jako całość.

Nie potrzeba specyfikować rozmiarów podczas alokacji, ani definiować jakiegokolwiek atrybutu. Atrybuty mają taką długość, jaka jest potrzebna podczas drukowania raportów, ale nie ogranicza to liczba danych, która można wprowadzić. W rzeczywistości można wprowadzać dane do rekordu bez definiowania atrybutów. Wszystkie nowoczesne bazy danych pozwalają na pewną elastyczność przy posługiwaniu się danymi, ale tylko niektóre dopuszczają to w takim zakresie jak PICK. Baza danych PICK umożliwia zamykanie dostępu do rekordów, ale nie transakcji (transakcja jest sekwencją uaktualniania bazy danych, która musi być traktowana niepodzielnie). Jeżeli nastąpi zaprzestanie wykonywania programu (ang. abort) przed zakończeniem transakcji, to częściowe zmiany nie zostaną wprowadzone. Istnieje specjalny rejestr, umożliwiający odtworzenie bieżącego stanu bazy danych w wypadku awarii dysku.

System wymaga regularnego tworzenia kopii plików bazy danych. W wypadku zniszczenia bazy danych, tracone są wszystkie dane, które zostały wprowadzone od ostatniego kopiowania. Tak prymitywne podejście do odtwarzania baz danych jest czymś normalnym, jeżeli chodzi o mikrokomputery, i na pewno nie jest zaletą PICKA.

PICK składa się z kilkunastu podsystemów. Interpreter poleceń (powłoka — ang. shell) nazywa się TCL (Terminal Control Language). ED jest edytorem pracującym w trybie wierszowym (nie ekranowym). Redagowanie tekstu prowadzi się podając numery wierszy. Zobaczenie wprowadzonych zmian w kontekście całego pliku jest możliwe tylko za pomocą listowania. Po wprowadzeniu, bądź usunięciu wiersza, użytkownik nie może wiele zdziałać bez otrzymania nowej numeracji wierszy od edytora. Nie dość, że użytkownik PICKA jest skazany na edytor wierszowy, to na dodatek najgorszy, jakiego kiedykolwiek używano.

PROC przetwarza polecenia wyłącznie z poleceniami interakcyjnymi jak ED, które są zapamiętywane w postaci procedur. Jest to złożony procesor posiadający cechy wejścia-wyjścia, sterujący wielodostępem, rozgałęzieniami (jak skoki i kontrole warunków), wyprowadzający na terminal znak gotowości systemu. Można go traktować jako język programowania na poziomie wydawania poleceń. Chociaż jego konstrukcja jest inna, to koncepcja przypomina powłokę UNIXA.

Język zapytań i generowania raportów nosi nazwę ACCESS. Posiada on możliwość selekcji, sortowania, dodawania, liczenia i listowania — zwyczajne możliwości, które występują także w złożonych systemach baz danych. W danym momencie można działać tylko na jednym pliku, nie można natomiast wykonywać operacji łącznych jak w relacyjnych bazach danych. Jeżeli, na przykład, istnieją dwa zbiory — zbiór zatrudnionych i zbiór sprzedaży, to w większości wypadków użycie systemu ACCESS nie dostarczy odpowiedzi na pytanie: kim są przełożeni akwizytorów, którzy nie zrealizowali swoich ofert. Wyjątek stanowi sytuacja, gdy identyfikatory wpisowy w jednym zbiorze są takie same jak w drugim. Gdy w zbiorze zatrudnionych i w zbiorze sprzedaży użyto identyfikatora zatrudnionego jako identyfikatora wpisu, wówczas jednym pytaniem można otrzymać listę wpisów, a następne pytanie spowoduje ich selekcję. Jeżeli stosuje się taką metodę przedstawiania zagadnień, to należy bardzo dokładnie zaprojektować

Hierarchia bazy danych PICK

Słownik systemu	— jeden dla systemu; zawiera nazwy kont, hasła i inne dane administracyjne.
Słownik użytkownika	— jeden dla konta (użytkownika); zawiera specyficzny dla danego użytkownika słownik poleceń i nazwy plików, jest widziany jak słownik dołączony do systemu.
Słownik plików	— wiele dla konta; zawiera definicje pól i relacje pomiędzy nimi.
Plik danych	— jeden lub wiele dla słownika danych; zawiera dane.
Wpis (ang. item)	— wiele dla pliku; zawiera dane dla jednego rekordu o długości do 32 767 bajtów; może być rekordem danych, programem źródłowym itp.; każdy wpis ma niepowtarzalny identyfikator; wszystkie pliki PICKA są dostępne przez algorytmy mieszające.
Atrybut	— wiele dla wpisu; odpowiada polu w rekordzie, linii tekstu itp.
Wartość	— jedna lub więcej dla atrybutu; może być użyta do natychmiastowego powielenia atrybutów rekordu.
Podwartość	— jedna lub więcej dla wartości; podobna do wartości, lecz działająca na poziomie niższym.

bazę danych, należy także upewnić się, że identyfikatory wpisów zostały odpowiednio dobrane.

PICK posiada także bardzo prymitywny formator tekstów, tzw. RUNOFF. W połączeniu z edytorem ED tworzy on godny początku lat sześćdziesiątych system redagowania tekstów. Jedynym językiem programowania dostępnym w PICKU jest Basic. Przypomina on nieco nieporęczny Microsoft Basic ale w rzeczywistości jest bardzo wygodnym językiem programowania. Do głównych cech tego Basica — niedostępnych w Basicu Microsofta — należą zmienne bez typów, napisy o długości do 32 KB, podprogramy mogące używać parametrów, wbudowane funkcje bazy danych, złożone strukturalnie instrukcje sterujące i brak numeracji linii.

Odwołanie się do wpisu w bazie danych następuje przez wczytanie do zmiennej napisowej, bądź zapis z tej zmiennej. Można odwoływać się do atrybutów, wartości i podwartości traktując napis jako trójwymiarową tablicę. Jeżeli — na przykład — w R znajdują się wpisy danych, to odwołanie R(4,3,2) odpowiada drugiej podwartości trzeciej wartości czwartego atrybutu. Takie wyrażenie może pojawić się po obu stronach wyrażenia arytmetycznego lub logicznego.

Instalowanie i użytkowanie PICKA

Instalacja PICKA jest dość łatwa. Ściąga się system z pierwszego dysku, a następnie korzystając z dyrektyw, które ukazują się na ekranie, ładuje się pozostałe 4 dyski. Nie trzeba nawet tworzyć stref (ang. partitions) na dysku stałym, gdyż PICK automatycznie odnajduje grupę wolnych cylindrów i organizuje je w strefę. Dlatego, gdy na tym samym dysku musi istnieć MS-DOS lub inny system operacyjny, PICKA należy instalować na końcu.

PICK jest jedynym znanym autorowi systemem operacyjnym, w którym podano jak połączyć kable, aby zainstalować odległy terminal. Co więcej, schemat połączeń jest poprawny. Niestety lista terminali, które można dołączyć jest krótka i wręcz dziwna (np. brakuje VT-100).

Kłopoty zaczynają się, gdy kończy działanie instrukcja instalacji systemu po załadowaniu ostatniego dysku. Na monitorze pojawia się znak gotowości systemu do przyjmowania poleceń, lecz operator nie ma żadnego sensownego planu dalszego działania. Po dwudziestominutowych poszukiwaniach właściwych poleceń dochodzi się do wniosku, że należy zarejestrowac się w systemie poleceniem SYSPROG (PICK przyjmuje tylko duże litery). Następnie można utworzyć konto za pomocą odpowiedniego polecenia CREATE-ACCOUNT.

W celach testowych autor utworzył plik PGM i napisał krótki program w Basicu (przy użyciu edytora ED), traktując go jako wpis pod nazwą TST1. Podczas próby skompilowania go pojawiła się informacja o złym formacie źródła. Po żmudnym wertowaniu podręcznika można znaleźć odpowiednią uwagę, że aby utworzyć plik (ang. program file), należy użyć edytora w celu „zmiany D-pointera w słowniku głównym na DC-pointer". Choć można mieć tylko mętne pojęcie, co to ma znaczyć, nie ma kłopotów z użyciem edytora do zredagowania głównego słownika i zamiany pointera. Nie wiadomo jednak, jaka idea temu wszystkiemu przyświeca. Wydaje się, że użytkownicy PICKA powinni używać języka PROC do utworzenia procedury CREATE-PROGRAM-FILE. Gdy ktoś długo pracuje z tym systemem, takie drobiazgi stają się niezauważalne, ale stykając się z PICKIEM po raz pierwszy można się łatwo do niego zniechęcić, gdyż nowy użytkownik powinien mieć jasny obraz pracy systemu.

PICK można ściągnąć z dysku stałego. System poświęca kilkanaście minut na tzw. „sprawdzanie węzłów". W tym czasie nie można używać klawiatury komputera, jakkolwiek odległe terminale są natychmiast pobudzane i można na nich od razu pracować. Nie należy jednak przypuszczać, aby modyfikacja bazy danych podczas jej sprawdzania była posunięciem bezpiecznym.

PICK nie odczytuje, ani nie zapisuje dysków elastycznych przygotowanych w systemie MS-DOS. Nie ma także dostępu do strefy systemu MS-DOS na dyskach stałych. Aby dostać się do systemu MS-DOS trzeba zakończyć pracę z PICKIEM i ściągnąć DOS. Powracając do PICKA trzeba znowu doczekać na weryfikację węzłów.

W czasie kilkudniowej pracy doświadczalnej PICK nigdy nie załamał się, jak również nie znaleziono w nim żadnych defektów. Wydaje się, że jest to solidne oprogramowanie.

Dokumentacja i funkcjonowanie PICKA

Dokumentacja jest kompletna, bardzo dobrze napisana i uzupełniona wieloma przykładami. Podręcznik zawiera bardzo dobry indeks, ale jest on tak schowany w opisie części serwisowej, że nowicjusz z pewnością go nie znajdzie. Istnieje również sekcja dyskowa, której zadaniem jest niesienie pomocy osobom stawiającym pierwsze kroki. Jest ona jednak tak uboga w informacje dotyczące PICKA, że trudno jest ocenić ją jako użyteczną.

Dla osób szczególnie zainteresowanych PICKIEM prowadzone są kursy, których program zakłada szczegółowe poznanie systemu. Kurs taki trwa tydzień i kosztuje ok. 900 dolarów. PICK jest tak bardzo różny od tradycyjnych systemów, że pożytek z kursu jest niewątpliwy.

Aby mieć jakiekolwiek pojęcie o funkcjonowaniu systemu PICK, porównano go z programem REVELATION, uruchamianym w systemie MS-DOS. REVELATION jest bazą danych przeznaczoną do współpracy z jednym użytkownikiem, wzorowaną na PICKU.

Uruchomiono program zapisujący 300 rekordów (każdy po 150 bajtów) do pliku początkowo pustego. Każdy rekord miał 20-bajtowy identyfikator. Po utworzeniu pliku postępowano dalej zgodnie z instrukcją i określono, iż potrzebne są 503 pozycje tablicy kluczy. Do wykonania tego programu PICK potrzebował 8 minut i 20 sekund. Rezultatem był plik składający się z 513 tys. bajtów. REVELATION do wykonania takiego samego programu (pomijając małą zmianę syntaktyczną) potrzebował 21 minut i 58 sekund; w tym czasie powstał plik składający się z 1 016 832 bajtów, zawierający ponadto wiele miejsca dla rekordów dodatkowych. Z tego względu nie można go porównywać z plikami PICKA.

Czas wykonania nie świadczy tu o wszystkim. Podczas pracy PICKA lampki kontrolne dysku rozbłyskiwały równomiernie, natomiast przy współpracy z REVELATION — światła kontrolne były przez większą część czasu zapalone i dał się słyszeć odgłos dysku poszukującego informacji. Dysk ów nigdy przedtem nie był poddawany takiemu egzaminowi.

Nie należy interpretować tych rezultatów w taki sposób, że PICK jest 3 razy szybszy niż REVELATION, a to ze względu na fakt, iż testowano tylko jeden program. Choć wiadomo, że PICK jest szybszy, trudno powiedzieć ile razy.

Przewaga jaką ma REVELATION nad PICKIEM polega na jego zintegrowaniu z systemem MS-DOS. Można czytać i zapisywać pliki, jak również wydawać polecenia systemem MS-DOS z wnętrza bazy danych. Ponadto, wiele innych elementów jest znacznie lepiej zaprojektowanych (np. istnieje edytor ekranowy). Wydaje się, że sensowne jest rozwijanie zastosowań przy użyciu bazy danych REVELATION, a następnie przeformatowanie opracowanego systemu na PICK, gdy jest gotów do eksploatacji.

* * *

Wiele mniej ważnych elementów systemu operacyjnego PICK zostało zaprojektowanych dosyć miernie, ale elementy zasadnicze są skonstruowane bardzo dobrze. Przede wszystkim można to powiedzieć o systemie plików oraz Basicu. PICK jest prosty i wszechstronny, wydaje się także efektywny oraz niezawodny. Wykonuje dobrze to, do czego został przeznaczony, choć wymaga nieco troskliwości, jeśli planuje się użyć go do początkowego zastosowania baz danych. Ze względu na wielodostęp, jest on prawdopodobnie najbardziej efektywnym (z uwagi na koszty) systemem działającym na PC/XT.

Opracował PIOTR STRUTYŃSKI

na podst. BYTE Vol. 10, No. 11, 1985

Tabak D.: Intel 80386 i nowe mikroprocesory 32-bitowe (1)

INFORMATYKA 1987, nr 1, s. 1

Charakterystyka rozwoju mikroprocesorów 16-bitowych rodziny Intel iAPX 86 oraz szczegółowe omówienie architektury mikroprocesora 32-bitowego Intel 80386.

Табак Д.: Интел 80386 и новые 32-разрядные процессоры (1)

INFORMATYKA 1987, № 1, с. 1

Характеристика развития 16-разрядных микропроцессоров из семейтсва Интел iAPX 86, а также подробное описание архитектуры 32-разрядного микропроцессора Интел 80386.

Gondzio M.: Sieci Petriego (1). Definicja i analiza sieci

INFORMATYKA 1987, nr 1, s. 5

Pierwsza część charakterystyki sieci Petriego. Podano definicję problemu i pojęć podstawowych, przykłady różnych rozwiązań i zastosowań sieci, a także analizę ich własności.

Гондзио М.: Сети Петри (1). Дефиниция и анализ сети

INFORMATYKA 1986, № 7—8, с. 5

Первая часть характеристики сети Петри. Приведена дефиниция проблемы и основных понятий, примеры разных решений и применений сети, а также анализ их свойств.

Cellary W.: Systemy wielomikroprocesorowe (2)

INFORMATYKA 1987, nr 1, s. 10

Dokończenie charakterystyki systemów wielomikroprocesorowych, obejmujące omówienie systemów z dzieloną magistralą lokalną oraz z dzieloną magistralą systemową.

Целлары В.: Многомикропроцессорные системы (2)

INFORMATYKA 1987, № 1, с. 10

Завершение характеристики многомикропроцессорных систем, включающей описание систем с раздельной местной общей шиной, а также с раздельной системной общей шиной.

Wrycza S.: Podejścia do tworzenia schematów konceptualnych

INFORMATYKA 1987, nr 1, s. 13

Omówienie trzech różnych sposobów podejścia do konstruowania schematów konceptualnych.

Врыча С.: Подход к созданию принципиальных схем

INFORMATYKA 1987, № 1, с. 13

Описание трех разных способов подхода к разработке принципиальных схем.

Bielecki J.: Język C — implementacja Lattice/Microsoft

INFORMATYKA 1987, nr 1, s. 17

Szczegółowe omówienie kompilatora języka C, opracowanego dla mikrokomputera IBM PC przez firmę Lattice i rozpowszechnianego również przez firmę Microsoft.

Белицкий Я.: Язык С — имплементация Lattice/Microsoft

INFORMATYKA 1987, № 1, с. 17

Подробное описание уомпилятора языка С, разработанного для микроЭВМ ИБМ РС фирмой Lattice и распространяемого фирмой Microsoft.

Ambroziak J.: Systemy ekspertowe (2)

INFORMATYKA 1987, nr 1, s. 19

Dokończenie charakterystyki systemów ekspertowych, zawierające porównanie z klasycznymi metodami przetwarzania informacji oraz omówienie motywów ich rozwoju i trudności realizacji.

Амброзяк Я.: Экспертные системы (2)

INFORMATYKA 1987, № 1, с. 19

Завершение характеристики экспертных систем, содержащей сравнение с классическими методами обработки данных, а также описание мотивов их развития и трудностей реализации.

Grabowicz R., Zalewski J.: System operacyjny PC-DOS (4)

INFORMATYKA 1987, nr 1, s. 21

Czwarta część charakterystyki systemu operacyjnego PC-DOS, zawierająca omówienie dodatkowych poleceń w wersji 3.0, możliwości konfigurowania systemu operacyjnego oraz funkcji standardowego interpretera poleceń.

Грабович Р.: Залевский Я.: Операционная система РС-ДОС (4)

INFORMATYKA 1987, № 1, с. 21

Четвертая часть характеристики операционной системы РС-ДОС, содержащей описание дополнительных команд в варианте 3.0, возможности создания разных конфигураций операционной системы и функций стандартной интерпретирующей программы команд.

Tabak D.: Intel 80386 and new 32-bit microprocessors (1)

INFORMATYKA 1987, No. 1, p. 1

Characteristics of the development of the Intel IAPX 16-bit microprocessor family und detailed discussion of architecture of Intel 80386 32-bit microprocessor.

Tabak D.: Intel 80386 und neue 32-bit-Mikroprozessore (1)

INFORMATYKA 1987, Nr 1, S. 1

Eine Charakteristik der Entwicklung von Intel iAPX 86 16-bit-Mikroprozessorfamilie und detaillierte Besprechung der Architektur von Intel 80386 32-bit-Mikroprozessor.

Gondzio M.: Petri nets (1). Definition and net's analysis

INFORMATYKA 1987, No. 1, p. 5

First part of Petri nets characteristics. Problem definition and basic terms, examples of different nets solutions and applications, as well as analysis of their features are presented.

Gondzio M.: Petri Netze (1). Definierung und Analyse der Netze

INFORMATYKA 1987, Nr. 1, S. 5

Erster Teil einer Charakteristik von Petri Netzen. Es wurden Definierung des Problems und der Grundbegriffe, Beispiele verschiedener Lösungen und Anwendungen der Netze, sowie Analyse ihrer Eigenschaften, angegeben.

Cellary W.: Multimicroprocessor systems (2)

INFORMATYKA 1987, No. 1, p. 10

Termination of the characteristic of the multimicroprocessor systems, which includes discussion of systems with shared local bus and with shared system bus.

Cellary W.: Mehrmikroprozessorsysteme (2)

INFORMATYKA 1987, Nr. 1, S. 10

Zweiter Teil einer Charakterstik von Mehrmikroprozessorsystemen, der eine Besprechung von Systemen mit geteilten Lokalmagistrale und mit geteilten Systemmagistrale umfasst.

Wrycza S.: Approaches to creating conceptual schemes

INFORMATYKA 1987, No. 1, p. 13

Presentation of the three different approaches to conceptual schemes constructing.

Wrycza S.: Einstelungen zur Konzeptualschemaentwicklung

INFORMATYKA 1987, Nr. 1, S. 13

Eine Besprechung von drei verschiedenen Entwicklungsmethoden für Konstruieren der Konzeptualschemen.

Bielecki J.: C-language — the Lattice/Microsoft implementation

INFORMATYKA 1987, No. 1, p. 17

Detailed discussion of the C-language compiler, which is elaborated for IBM PC by Lattice and is distributed by Microsoft too.

Bielecki J.: C-Sprache-Implementierung von Lattice/Mikrosoft

INFORMATYKA 1987, Nr. 1, S. 17

Eine detaillierte Besprechung von C-Sprache-Kompilierer, der für IBM PC von der Firma Lattice erarbeitet und auch von der Firma Microsoft verbreitet wird.

Ambroziak J.: Expert systems (2)

INFORMATYKA 1987, No. 1, p. 19

Second part of expert systems characteristics, which includes a comparison with classic information processing systems and discussion of its development motivation, as well as realization dificulties.

Ambroziak J.: Expertensysteme (2)

INFORMATYKA 1987, Nr 1, S. 19

Zweiter Teil einer Charakteristik von Expertensystemen, der ein Vergleich mit klassischen Methoden der Informationsverarbeitung, sowie eine Besprechung der Entwicklungsmotivierung und Realisationsschwierigkeiten umfasst.

Grabowicz R., Zalewski J.: PC-DOS operating system (4)

INFORMATYKA 1987, No. 1, p. 21

Fourth part of the PC-DOS operating system characteristics, which includes discussions of additional commands in the 3.0 verision, possibilities of operating system configurating, as well as functions of the standard command interpreter.

Grabowicz R., Zalewski J.: PC-DOS-Betriebssystem (4)

INFORMATYKA 1987, Nr 1, S. 21

Vierter Teil einer Charakteristik von PC-DOS-Betriebsystem, der eine Besprechung von zusätzlichen Befehlen in der 3.0 Version, Möglichkeiten der Betriebssystemkonfigurierung, sowie Funktionen des Standardbefehlinterpreters umfasst.

Terminologia lokalnych sieci komputerowych (2)

Po przedstawieniu lokalnych sieci komputerowych na tle modelu odniesienia ISO, w rozważaniach dotyczących używanej w tej dziedzinie terminologii warto zatrzymać się nad ogólnymi właściwościami tych sieci. Do podstawowych cech charakteryzujących sieci lokalne należą: **konfiguracja, ośrodek transmisji, rodzaj transmisji** i **metoda dostępu do ośrodka.**

W krajowych opracowaniach technicznych na temat sieci lokalnych konfigurację nazywa się topologią — za angielskim określeniem topology. Jest to typowy przykład niewłaściwego i piętnowanego przez językoznawców sposobu przejęcia terminu z języka obcego. Wyraz topologia nie ma w języku polskim takiego znaczenia, w jakim jest użyty w języku angielskim, tj. — układ połączeń poszczególnych jednostek pełniących samodzielne funkcje. Dlatego, mając inne polskie odpowiedniki na oznaczenie tego pojęcia wahałbym się, czy wprowadzić nowe znaczenie tego wyrazu do języka polskiego, tym bardziej, że w języku informatycznym na oznaczenie układu powiązanych urządzeń komputerowych używa się wyrazu konfiguracja.

W języku potocznym wyraz **konfiguracja** znaczy także tyle, co „ukształtowanie, układ" (Słownik wyrazów obcych PWN. Wydanie pierwsze, 1978). Nie ma więc logicznego powodu, aby mówić „topologia sieci", skoro równie zrozumiały i znaczący dokładnie to samo jest zwrot „konfiguracja sieci". Do najbardziej rozpowszechnionych konfiguracji sieci lokalnych należą: **konfiguracja magistralowa** (ang. bus topology) i **konfiguracja pierścieniowa** (ang. ring topology). Oprócz tego, teoretycznie możliwych jest wiele inych układów połączeń, tzn. konfiguracji, np.: **gwiaździsta** (ang. star), **drzewiasta** (ang. tree), **siatkowa** (ang. grid) i in. Niektórzy znawcy przedmiotu potrafią nawet określić subtelną różnicę między konfiguracją pierścieniową a pętlową (ang. loop). Spotkałem się, mianowicie, z rozróżnieniem, że konfigurację pierścieniową tworzą stacje, których odłączenie od sieci powoduje przerwanie jej zamkniętej struktury, natomiast odłączenie stacji od pętli nie powoduje naruszenia jej zamkniętej struktury.

Drugą cechą charakterystyczną sieci lokalnych prowadzącą do rozróżnienia sieci o podobnej konfiguracji jest środowisko fizyczne, materiał użyty do połączenia elementów sieci i przesyłania wiadomości, zwany po angielsku transmission medium. Podobnie jak w wypadku wyrazu topologia, próbuje się używać polskiego wyrazu medium w znaczeniu, które ten wyraz ma w języku angielskim. Choć taka praktyka bywa stosowana w informatyce, np. w przypadku terminu port, a ponadto wyrazu medium używa się niekiedy w zbliżonym znaczeniu w fizyce (ośrodek, środowisko, w którym zachodzi dane zjawisko — Słownik wyrazów obcych PWN. Wydanie pierwsze, 1968), nie sądzę, aby było to konieczne w dziedzinie sieci lokalnych, skoro w polskim słownictwie telekomunikacyjnym używa się od dawna wyrazu **ośrodek** (ośrodek transmisji). **Ośrodki transmisji** w lokalnych sieciach komputerowych w zasadzie ograniczają się do trzech rodzajów: **pary skręcane** (ang. twisted pairs), **kable współosiowe** (kable koncentryczne, ang. coaxial cables) i **światłowody** (ang. fiber optics, włókna światłowodowe).

Znając możliwości doboru konfiguracji sieci i użycia odpowiedniego ośrodka transmisji, warto zastanowić się nad nazewnictwem dotyczącym kolejnej cechy charakteryzującej sieci lokalne, odnoszącej się do rodzaju transmisji na poziomie przekazywania sygnałów fizycznych. Obecnie stosowane środki techniczne umożliwiają w zasadzie dwa rodzaje transmisji: **transmisję w pasmie podstawowym** (ang. baseband system, baseband signalling) i **transmisję szerokopasmową** (ang. broadband system). Według definicji zawartych w normach sieci Ethernet (CSMA/CD) **transmisja w pasmie podstawowym** polega na bezpośrednim przesyłaniu do ośrodka sygnału niezmodulowanego, tak że w dowolnej chwili w ośrodku istnieje tylko jeden sygnał informacyjny (wykluczając kolizje). **Transmisja szerokopasmowa**, natomiast polega na przesyłaniu do ośrodka zmodulowanego sygnału informacyjnego, przenoszonego przez falę nośną, wskutek czego w tej samej chwili w ośrodku może istnieć wiele nie zakłócających się wzajemnie sygnałów informacyjnych, zajmujących odpowiednie pasma w pełnym widmie częstotliwościowym. Jak wynika z tego bardzo poglądowego opisu, znacznie łatwiejsze do realizacji i tańsze są systemy transmisji w pasmie podstawowym, dlatego obecnie zdecydowana większość sieci lokalnych jest realizowana tą techniką.

Dokonawszy wyboru konfiguracji sieci, ośrodka transmisji i rodzaju transmisji, projektant lokalnej sieci komputerowej staje przed najważniejszym bodaj problemem doboru tzw. metody dostępu do ośrodka transmisji, tj. logicznej zasady przydzielania ośrodka transmisji ubiegającym się o niego stacjom. W sieciach lokalnych zasada przydziału jest na ogół zdecentralizowana, tzn. nie ma jednej, nadrzędnej jednostki decydującej o przydziale według założonego algorytmu. Największą popularność zdobyły dotąd dwie metody dostępu o następujących nazwach angielskich:

— carrier-sense multiple access with collision detection (CSMA/CD), co można określić po polsku jako wrażliwy na falę nośną dostęp wspólny z wykrywaniem kolizji (odpowiednie człony tego określenia omówiono w poprzednim numerze Informatyki),

— token access procedure, co oznacza sposób zarządzania użyciem współdzielonego ośrodka, polegający na wykluczaniu wzajemnym na podstawie posiadania symbolu uprawnienia. Nazwy drugiej metody nie można przetłumaczyć na język polski nie znając odpowiednika wyrazu token, który oznacza symbol uprawnienia przekazywany między stacjami w celu wskazania, która stacja ma aktualną kontrolę nad ośrodkiem. Z terminologicznego punktu widzenia dobór polskiego odpowiednika terminu token jest więc istotnym problemem. W polskiej literaturze fachowej zetknąłem się dotychczas z kilkoma różnymi odpowiednikami tego terminu, m.in.: znacznik, wyznacznik i żeton. Choć w języku potocznym wyraz token można tłumaczyć na znacznik, to w informatyce termin znacznik jest już obciążony wieloma innymi znaczeniami i nie sądzę, aby jego przyjęcie w tym wypadku było najlepszym rozwiązaniem. Z kolei, przyjęcie wyrazu wyznacznik, wobec całkowicie odmiennego znaczenia, jakie ten termin ma w dziedzinie pokrewnej, tj. w matematyce, uważam za nieporozumienie. Za najlepszy z trzech cytowanych terminów, odpowiedników wyrazu token, wypada więc uznać wyraz żeton, choć ma on zabarwienia znaczeniowe nie wskazujące na specjalne uprawnienia i jest wyrazem obcego pochodzenia. Sądzę, że wyrazem dokładnie określającym znaczenie terminu token (symbol uprawnienia), a jednocześnie pozbawionym innych obciążeń znaczeniowych — oraz rdzennie polskim — jest wyraz **znamię**, który można by przyjąć za odpowiednik terminu angielskiego. Wówczas można by mówić i pisać — **dostęp znamieniowy** (ang. token access), **metoda dostępu znamieniowego** (ang. token access procedure) itd.

JANUSZ ZALEWSKI

KONIEC TWOICH PROBLEMÓW

Nareszcie wszystkie potrzebne Ci dane dotrą na czas w przejrzystej formie tabel i wykresów.

Pakiet oprogramowania na mikrokomputery 16-bitowe

MEGA — BANK

to nowe MEGA możliwości jakie otwierają się przed Twoim przedsiębiorstwem.

Wyobraź sobie

MILIARD

rekordów, które możesz zapełnić według własnych potrzeb i uznania.

Miliard kooperantów, miliard pracowników, miliard produktów — z tym wszystkim nasz MEGA-BANK poradzi sobie bez trudu.

System jest łatwy w obsłudze i opracowany w języku polskim.

Gwarantujemy satysfakcję!

COMPUTER STUDIO KAJKOWSCY

PROFESJONALNE OPROGRAMOWANIE MIKROKOMPUTERÓW

ul. Balladyny 3B, 81-524 Gdynia, tel.: 29-00-18, 24-01-50

EO/74/KJ/87